Оглавление

Содержание

Интерфейсы и распределенные вычисления

ЧАСТЬ III

Глава

15

Модули DLL и подключаемые компоненты

Джо Бонавита
Джин Паризье

Предположим, что вас пригласили принять участие в разработке приложения, которое будет обрабатывать информацию из базы данных и формировать отчеты. И вам, к примеру, приходится заниматься сразу двумя приложениями, которые создают отчеты одинакового формата. Вряд ли кому-нибудь в такой ситуации не придет в голову идея создать один модуль формирования отчетов и вызывать его из обоих приложений, причем каждое из них будет передавать в этот модуль свои данные. Так можно было бы и поступить, да есть одна загвоздка — все в этом мире переменчиво. В один прекрасный день раздается звонок и один из заказчиков просит внести в отчет еще данные из двух-трех полей (Ничего, никогда не знаешь, что придет в голову заказчику завтра). Скорее всего вы подумаете "такое уже не раз случалось. Придется кое-что изменить в модуле формирования отчета, перекомпилировать оба проекта и разослать новые версии заказчикам". Было бы еще лучше, если бы вообще не нужно было трогать проект. Если модуль формирования отчета оформлен в виде *динамически связываемой библиотеки* (DLL-модуль), тогда можно было бы внести необходимые изменения только в этот модуль и только его рассылать пользователям. Все остальные компоненты проекта при этом можно оставить без изменения.

Конечно, если речь идет всего о двух проектах, то, в конце концов, повторно скомпилировать их большого труда не представляет, а вот если таких проектов 3, 4, 5 или 10, то объем работы уже заставляет почесать затылок.

В этой главе мы и остановимся на том, как с помощью C++Builder создавать модули DLL, экспортировать функции и классы, загружать модули DLL статически и динамически, размещать в них экранные формы однодокументного (SDI-) и многодокументного (MDI-) приложений. Будет также показано, чем отличаются модули DLL от пакетов. Мы сформируем базовый пакет, будем экспортировать его функции, загружать и выгружать их. В завершение мы рассмотрим, как использовать в C++Builder-проекте модули DLL, созданные в среде Visul C++, и наоборот — в проекте Visul C++ использовать DLL, созданные с помощью C++Builder.

Использование мастера DLL Wizard

Работая в среде C++Builder, проще всего создать модуль DLL с помощью мастера ***DLL Wizard***. Для того чтобы вызвать его на экран, выберите команду File⇨New, в затем укажите DLL Wizard на вкладке New и щелкните на OK. В окне мастера можно выбрать несколько опций (рис. 15.1).

- C. При выборе этого переключателя флажок Use VCL (использовать VCL) блокируется. Оно и понятно — компоненты из библиотеки VCL можно включать только в проект на языке C++. Программе на языке C незнакомы такие понятия, как класс, объект и т.п., а потому, если в создаваемом модуле вы собираетесь использовать программные компоненты, созданные на языке C++, переключатель C устанавливать не следует.
- C++. Выбор этой опции позволяет использовать любые компоненты из библиотеки VCL, организовать многопотоковый режим выполнения и создавать DLL в стиле Visul C++. При выборе этой опции для построения выполняемого модуля DLL будет использован компилятор языка C++, а значит, в текст программы можно включать любой программный код на этом языке.
- Use VCL. Если этот флажок установлен, то в состав DLL можно включать компоненты из библиотеки VCL. Выше уже отмечалось, что этот флажок блокируется, если выбран переключатель C. При установке флажка Use VCL C++Builder включает директиву `#include <VCL.h>` в главный модуль программного кода DLL и соответственно настраивает код запуска и опции компоновщика. Обращаю ваше внимание на то, что

при установке этого флажка соседний флажок — Multi Threaded — блокируется и его нельзя сбросить. Причина в том, что компоненты VCL нуждаются в поддержке многопотокового режима выполнения программы.

- Multi Threaded. Установка этого флажка задает поддержку многопотокового режима выполнения программы DLL. Рекомендуется, если нет особых возражений, всегда устанавливать этот флажок. Если установлен флажок Use VCL, то установку Multi Threaded среда C++Builder выполнит автоматически, причем после этого флажок блокируется и сбросить его нельзя.
- VC++ Style DLL. Установка/сброс этого флажка задает тип точки запуска создаваемого модуля DLL. Если необходимо, чтобы при запуске вызывалась функция `DLLMain()`, как то предусмотрено в Visual C++, установите флажок VC++ Style DLL; в противном случае будет вызываться функция `DLLEntryPoint()`. Как правило, этот флажок сбрасывается, причем независимо от того, планируется ли в дальнейшем использовать этот модуль DLL совместно с приложением, разработанным в среде Visual C++.

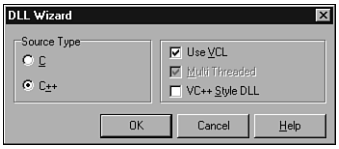


Рис. 15.1. Диалоговое окно мастера DLL Wizard с настройками, установленными по умолчанию

Чаще всего при работе с матером используются настройки, предлагаемые по умолчанию. Но иногда такой вариант по каким-либо причинам не подходит и нужно вспоминать, что влечет за собой выбор той или иной опции. Более подробную информацию об этих опциях можно почерпнуть из системы оперативной справки C++Builder.

Разработка и использование DLL

Использование модулей DLL (Dynamic Link Library — динамически связываемая библиотека) — вероятно, один из самых распространенных в настоящее время способов создания модульных приложений. DLL представляет собой особый вид выполняемого кода, который загружается в память приложением в процессе его выполнения и может содержать такой же код, как и обычный выполняемый файл программы. Программу, оформленную в виде DLL, нельзя запустить на выполнение в одиночку — кто-то ее должен загрузить и "толкнуть". Этим "кто-то" может быть основной выполняемый файл приложения или другой модуль DLL.

Модуль DLL можно загружать статически или динамически. Статически загружаемые модули связываются с главной выполняемой программой при ее компоновке и остаются загруженными в память все время, пока выполняется приложение. Динамически загружаемый модуль в процессе работы приложения при необходимости можно выгрузить из памяти, освободив таким образом ресурсы. Другое существенное отличие между статически и динамически загружаемыми DLL в том, что без первых приложение нельзя запускать на выполнение, а модуль второго типа вообще может отсутствовать на компьютере, если приложение выполняет такие операции, в которых этот модуль "не участвует". Работа приложения прекратится только в том случае, если потребуется обращение к функции, размещенной в отсутствующем модуле DLL.

Но DLL отличаются не только способом загрузки, но и содержимым. Чаще всего в модуле DLL присутствует программный код и средства поддержки экранных форм. Но встречаются и такие DLL, которые содержат ресурсы — пиктограммы, растровые изображения и т.п. Распространена также практика помещать в DLL таблицы строковых констант — это позволяет легко адаптировать приложение к потребностям пользователей, говорящих на разных языках.

Приступая к разработке DLL-модуля, нужно прежде всего решить, что именно в этом модуле должно быть доступно другим компонентам приложения — будут ли это отдельные функции или классы. Если предполагается экспортировать функции, то настоятельно рекомендуется, хотя это и необязательно, разработать экспортируемую функцию таким образом, чтобы она не обращалась к функциям из других модулей. В противном случае придется распространять оба модуля комплектно.

Обычно в состав модулей DLL входят функции оболочки, функции работы с сетью, какие-либо специальные классы, например классы, ориентированные на использование OpenGL.

Статическое связывание DLL

В следующих трех разделах этой главы мы займемся созданием приложения, которое будет обращаться к DLL-модулям тремя разными способами. Все файлы этого приложения с незатейливым именем `CallingApp` (Calling Application — вызывающее приложение) вы можете скопировать из папки `source\Chapter15\CallingApp` на прилагаемом к этой книге компакт-диске.

Сначала будет показано, как создать и подключить к приложению статически связываемый модуль DLL. Файлы этого модуля находятся в папке `source\Chapter15\SimpleDLL` на прилагаемом к книге компакт-диске. Затем будет рассмотрен процесс создания динамически связываемого модуля DLL, файлы которого находятся в папке `source\Chapter15\DynamicDLL` на том же компакт-диске. Заключительный этап работы — создание пакета для нашего приложения. Файлы пакета находится в папке `source\Chapter15\Package` на все том же прилагаемом к книге компакт-диске.

В этих трех разделах будет показано, как создавать каждый компонент и вносить соответствующие изменения в основные файлы приложения Calling Application, чтобы на каждом этапе приложение могло выполнять определенные функции. Начнем работу с создания версии приложения, которая будет пользоваться “услугами” статически связываемого модуля DLL.

Создайте новый проект и установите в свойстве `Caption` экранной формы нового приложения значение **`Calling Application`**. Сохраните этот модуль в файле `CallingForm.cpp`, а файлу проекта присвойте наименование `CallingApp.bpr`. Этот проект мы будем модифицировать по мере изучения материала этой главы, а законченный проект имеется на прилагаемом к книге компакт-диске.

Выберите команду View⇨Project Manager в главном меню среды C++Builder — на экране появится диалоговое окно Project Manager (рис. 15.2). Щелкните на пиктограмме New — на экран будет выведено диалоговое окно хранилища объектов Object Repository. Выберите на вкладке New пиктограмму DLL Wizard и щелкните на OK. После этого на экране откроется окно мастера DLL Wizard.

В диалоговом окне мастера не меняйте предлагаемых по умолчанию опций и сразу же щелкните на кнопке OK. После этого выберите в главном меню C++Builder команду File⇨Save All и сохраните файл модуля DLL под именем `SimpleDLL.cpp`, файл проекта — под именем `SimpleDLL.bpr`, а имя файла группы проектов задайте как `DLLProjectGroup.bpg`. В главном меню C++Builder опять выберите команду File⇨New. На сей раз на вкладке New выберите Unit и щелкните на кнопке OK. Сохраните файл этого модуля под именем `DllFunctions.cpp`. Именно в этом файле будет храниться программный код DLL. Вы навер-

няка уже обратили внимание на то, что файлам проекта и первого модуля DLL мы присвоили одинаковые имена. В отличие от того, что происходит при создании нового приложения, C++Builder самостоятельно не создает файл проекта или главного модуля реализации с теми же именами, что и имя проекта. Это происходит по той простой причине, что DLL не нуждается в какой-либо экранной форме (хотя никто и не запрещает помещать ее в DLL, если в этом есть необходимость). DLL может использовать файл, в котором содержится функция `DllEntryPoint()`. Она играет в DLL ту же роль, что и функция `WinMain()` в обычной Windows-программе.

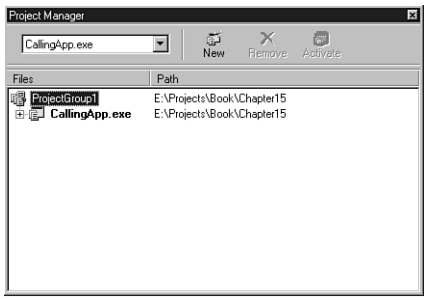


Рис. 15.2. Проект `CallingApp` в диалоговом окне Project Manager

В окне Project Manager можно открывать одновременно несколько проектов, причем для каждого из них выводится древовидный список всех файлов, включенных в состав проекта.

Добавьте в файл `DllFunctions.cpp` следующий программный код функции `Say()`:

```
void Say(char *WhatToSay)
{
    ShowMessage("This is from within the DLL\ n" +
               // "Это сообщение от модуля DLL\ n"
          (String)WhatToSay);
}
```

В файл `DllFunctions.h` добавьте объявление этой функции:

```
extern "C" void __declspec(dllexport) Say(char *WhatToSay);
```

Это объявление сообщает C++Builder, что функция `Say()` будет экспортируемой. Ключевое слово `extern` означает, что C++Builder должен использовать соглашение о вызове, принятое в языке C, и любой модуль, созданный с помощью компилятора, который поддерживает экспорт этого типа, сможет импортировать эту функцию. Спецификатор класса памяти `dllexport` позволяет экспортировать функции и объекты, определенные с этим спецификатором, и используется для совместимости с диалектами Microsoft C/C++. Кроме того, использование спецификатора `dllexport` позволяет обойтись без файла определений (эти файлы имеют расширение `.def`). Если используется модуль DLL, созданный с помощью другого компилятора, то, возможно, потребуется вместе с ним использовать и `def`-файл. Ниже вы увидите, что мы будем использовать и спецификатор класса памяти `dllimport`. Он применяется из тех же соображений, что и `dllexport`: использование этого спецификатора обеспечивает совмес-

тимость и позволяет импортировать функции, данные и объекты из других DLL. Более подробную информацию вы можете получить в системе оперативной справки C++Builder.

Теперь сохраните все открытые файлы и проекты и приступим к "наполнению" DLL чем-то полезным. Убедитесь, что проект SimpleDLL является текущим активным — имя активного проекта выводится в строке заголовка главного окна C++Builder. Этот заголовок должен выглядеть так: C++Builder 5 - SimpleDll.

Если же активным является другой проект, настройку среды можно изменить множеством способов. Проще всего это сделать с помощью диалогового окна Project Manager. (Чтобы вывести его на экран выберите в главном меню команду View⇨Project Manager.) В этом диалоговом окне дважды щелкните на имени того проекта, который собираетесь сделать активным, или выберите это имя и щелкните на кнопке Activate в правом верхнем углу окна. Есть и еще один способ — щелкнуть на имени облюбованного проекта правой кнопкой мыши и выбрать в контекстном меню команду Activate.

Текущий активный проект после этого можно компилировать — C++Builder создаст для DLL файл библиотеки с расширением .lib и выполняемый файл с расширением .dll.

Вернитесь теперь в приложение Calling Application и добавьте в его экранную форму CallingForm объект компонента TButton (кнопку). Присвойте свойству Caption этого объекта значение **Static**. В метод обработки события OnClick нового объекта добавьте следующий программный код:

```
void __fastcall TForm1::Button1Click(TObject *Sender)
{
    // Вызывается функция из модуля DLL.
    Say("Hello");
}
```

И последняя операция — выберите команду Project⇨Add to Project. Измените в поле Files of Type значение на **Library File (*.lib)** и выберите в списке файл SimpleDll.lib. Именно этот файл C++Builder будет использовать при связывании вызовов функций, размещенных в DLL. В файл CallingForm.cpp нужно добавить ссылку на файл заголовка DLL. Добавьте в него следующую строку:

```
#include "DllFunctions.h"
```

Теперь сохраните проект CallingApp и заново скомпилируйте оба проекта — и CallingApp, и SimpleDLL. Для этого в главном меню нужно выбрать команду Project⇨Build All Projects. Убедитесь, что активным в среде по-прежнему является проект CallingApp, и запустите его на выполнение. Щелкните на единственной кнопке, которая имеется в диалоговом окне приложения, и на экране появится окно сообщения, как показано на рис. 15.3.

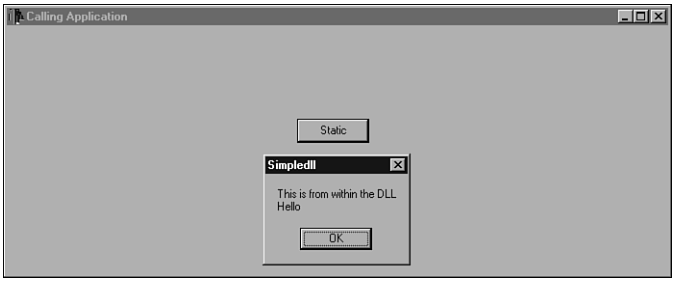


Рис. 15.3. В окне приложения CallingApp выводится сообщение, сформированное модулем DLL

Итак, все, что нужно сделать для использования в приложении статически связанного модуля DLL, — это включить в файлы `.cpp` приложения директивы включения файла заголовка DLL, а в состав проекта включить соответствующий `.lib`-файл. Все остальное выполняется так же, как и при работе с обычными модулями программного кода.

Импортирование функций из динамически связанных модулей DLL

Для динамического связывания модуля DLL с основным приложением нужно выполнить следующие операции.

- Загрузить DLL и получить указатель на этот модуль.
- Извлечь указатель на размещенную в DLL функцию, которую планируется вызывать.
- Вызвать функцию.
- Освободить память, занятую модулем DLL.

Существует два способа импортирования функций из динамически связанного модуля DLL.

- Создать с помощью `typedef` новый тип, используя прототипы экспортируемых функций.
- Выполнять приведение типа при каждом вызове `GetProcAddress()`.

Для того чтобы остановиться на том или другом варианте, нужно посмотреть не имеют ли все интересующие вас функции одинаковый формат прототипа (тип возвращаемого значения и типы аргументов). Если это так, то можно воспользоваться первым из указанных способов. Функция Windows API `GetProcAddress()` возвращает указатель на экспортируемую функцию, размещенную в DLL, принимая в качестве аргументов указатель на загруженный модуль DLL и имя той функции, на которую требуется получить указатель. Пусть, например, одна экспортируемая функция в DLL имеет прототип `int Add(int x, int y)`, а другая — `int Subtract(int x, int y)`. В таком случае можно использовать одно и то же приведение типов для обоих функций.

```
typedef (int (__import *AddSubtract)(int, int))
        AddSubtract *MyAdd, *MySubtract;
MyAdd  = (AddSubtract *)GetProcAddress(Dll, "_Add");
MySubtract = (AddSubtract *)GetProcAddress(Dll, "_Subtract");
```

Обратите внимание на то, что первым аргументом `GetProcAddress()` является переменная `Dll`. Эта переменная содержит указатель на модуль DLL, который загружен ранее с помощью функции Windows API `LoadLibrary()`. О механизме динамической загрузки модуля DLL мы поговорим потом, когда будем рассматривать пример использования динамического связывания.

В первой строке приведенного выше фрагмента программы объявляется новый тип `AddSubtract` с помощью спецификатора класса памяти `typedef`. Этот новый тип фактически "подменяет" собой спецификацию `int (__import *)(int, int)`. Такой тип имеет функция, которая возвращает значение `int` и принимает два аргумента типа `int`. Ключевое слово `import` сообщает компилятору C++Builder, что функция будет импортирована из внешнего источника. Вместо `__import` можно было бы подставить `__declspec(dllimport)`.

Во второй строке программы объявляются две переменные, имеющие тип указателя на новый тип `AddSubtract`.

В списке аргументов вызова GetProcAddress() перед именами функций нужно вставлять символы "подчеркивание", поскольку C++Builder добавляет их автоматически в имена всех функций, экспортируемых из DLL.

В третьей и четвертой строках этим переменным присваиваются значения, возвращаемые функцией GetProcAddress(). Обратите внимание, что в этих операторах выполняется приведение типа значения, возвращаемого GetProcAddress(), к AddSubtract. Использование нового типа избавляет программиста от необходимости использовать более длинную спецификацию при приведении типа.

В противном случае переменные MyAdd и MySubtract нужно объявить следующим образом:

```
int(__import *MyAdd)(int x, int y);
int(__import *MySubtract)(int x, int y);
```

Вызов функции GetProcAddress() будет выглядеть следующим образом:

```
MyAdd = (int(__import *)(int, int))GetProcAddress(Dll, "_Add");
MySubtract = (int(__import *)(int, int))GetProcAddress(Dll,
      "_Subtract");
```

Конечно, первый вариант более предпочтителен с точки зрения "читабельности" текста программы, но он хорошо работает только в том случае, если все экспортируемые функции в DLL аналогичны по типу возвращаемого значения и типам аргументов. Если же в приложении импортируется только одна функция из DLL или в DLL имеется большое разнообразие форматов экспортируемых функций, то второй вариант методики приведения типов вполне приемлем.

В принципе, выбор того или иного варианта — дело вкуса программиста. Лично я предпочитаю первый вариант, но многие из моих коллег пользуются вторым, и это не мешает им создавать вполне работоспособные программы. Поработав некоторое время над созданием приложений, импортирующих функции из DLL, вы найдете приемлемый для себя стиль. Возможно, вам понравится идея создать с помощью директивы #define специальный макрос, который и будет заниматься приведением типов. В тех примерах, которые мы будем рассматривать далее в этой главе, я использую методику на основе typedef.

Теперь займемся созданием нового модуля DLL, который будет динамически связываться с приложением. Выберите в меню команду File⇨New, а затем на вкладке New выберите DLL Wizard. В диалоговом окне мастера не меняйте предлагаемых по умолчанию опций и сразу же щелкните на кнопке OK. Сохраните созданный C++Builder файл модуля DLL под именем DynamicDll.cpp, а файл проекта — под именем DynamicDll.bpr. Снова выберите команду File⇨New, но на вкладке New выберите Unit. Сохраните модуль под именем DynamicFunctions.cpp.

Этот проект можно сохранить в той же папке, что и проект SimpleDLL, либо в отдельной папке. Если для проекта будет выбрана отдельная папка, то для загрузки созданного в этом проекте модуля DLL придется указывать полный путь к нему.

Откройте файл DynamicFunctions.cpp и добавьте в него следующий фрагмент:

```
void DynamicSay(char *WhatToSay)
{
    ShowMessage("This is from within the Dynamic DLL\ n" +
    // "Это сообщение от динамически загруженного модуля DLL\ n"
          (String)WhatToSay);
}
```

В файл заголовка также добавить строку:

```
extern "C" void __declspec(dllexport) Say(char *WhatToSay);
```

Сохраните проект и скомпилируйте его.

Вновь откройте проект `CallingApp` и добавьте в его экранную форму еще одну кнопку (объект компонента `TButton`). В свойстве `Caption` объекта установите значение **`Dynamic`**. Компоновка элементов управления на поле экранной формы должна выглядеть примерно так, как на рис. 15.4.

Рис. 15.4. Компоновка элементов управления в главном окне приложения `CallingApp`

В файл `CallingForm.h` добавьте строку определения нового типа в соответствии с форматом прототипа экспортируемой функции DLL. Там же объявите новую переменную, имеющую тип указателя на новый тип.

```
private:
    typedef void __declspec(dllimport) SayType (char *);
    SayType *LoadSayFunction;
```

Добавьте приведенный в листинге 15.1 метод обработки события `OnClick` новой кнопки.

Листинг 15.1. Обработчик события `Button2Click`

```
void __fastcall TForm1::Button2Click(TObject *Sender)
{
    // Сначала загружается модуль Dll.
    HINSTANCE Dll = LoadLibrary("Dynamic.dll");

    // Проверка, выполнена ли загрузка.
    if (Dll)
    {
        // Получить адрес функции.
        LoadSayFunction = (SayType *)GetProcAddress(Dll, "_Say");
        if (LoadSayFunction)
            LoadSayFunction("Dynamic Hello");
        else
            ShowMessage(SysErrorMessage(GetLastError()));

        // Удаление модуля DLL из памяти.
        FreeLibrary(Dll);
    }
    else
```

```
    {
        ShowMessage(SysErrorMessage(GetLastError()));
        ShowMessage("Unable to load the DLL");
                  // "Модуль DLL не загружен!"
    }
}
```

Приведенный текст программы нуждается в кратких пояснениях.

Обратите внимание на первый оператор:

```
HINSTANCE Dll = LoadLibrary("Dynamic.dll");
```

`HINSTANCE` — тип значения, возвращаемого функцией `LoadLibrary()`. Эта функция входит в состав Win32 API и более подробную информацию о ней можно получить в документе *Win32 SDK Reference*. Функция загружает модуль DLL, имя которого передается ей в качестве аргумента. В принципе, нужно включать в аргумент полный путь к DLL-файлу, но если модуль находится в одном из каталогов, которые Windows просматривает по умолчанию, можно ограничиться только именем собственно файла. Если не удается загрузить DLL-модуль, `LoadLibrary()` возвращает `NULL`. Вот почему в следующем операторе анализируется значение переменной `Dll`:

```
If (Dll)
```

Если окажется, что модуль DLL не загружен (переменная `Dll` имеет значение `NULL`), то выполняются операции в блоке `else`:

```
 else
 {
     ShowMessage(SysErrorMessage(GetLastError()));
     ShowMessage("Unable to load the DLL");
               // "Модуль DLL не загружен!";
 }
```

В результате на экран будет выведено сообщение об ошибке.

Если же загрузка прошла успешно, нужно получить указатель на функцию, экспортируемую модулем DLL (адрес точки вызова функции). В составе Win32 API имеется функция `GetProcAddress()`, которая принимает два аргумента: указатель на загруженный модуль DLL и имя импортируемой из этого модуля функции. `GetProcAddress()` возвращает значение типа `FARPROC`, которое можно привести к любому типу. В данном случае выполняется приведение к типу `SayType`.

```
LoadSayFunction = (SayType *)GetProcAddress(Dll, "_Say");
```

Теперь адрес функции `_Say()` находится в переменной `LoadSayFunction` и с ней можно обращаться как с именем функции.

При создании экспортируемой функции в составе DLL-модуля среда C++Builder по умолчанию добавляет символ "подчеркивание" перед заданным в тексте программы именем функции. Но имеется возможность отключить такой режим преобразования имени. Нужно открыть диалоговое окно Project Options (с помощью команды меню Project⇨Options) и на вкладке Advanced Compiler сбросить флажок Generate Underscores (рис. 15.5).

Перед тем как пользоваться полученным указателем, нужно для страховки проверить, не вернула ли функция `GetProcAddress()` по каким-либо причинам значение `NULL`. Если этого не случилось, переменную, содержащую указатель, можно использовать в программе как имя обычной функции:

```
LoadSayFunction("Dynamic Hello");
```

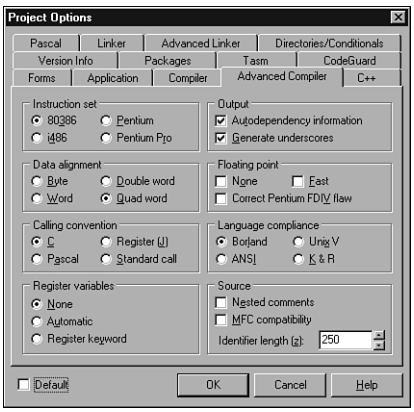


Рис. 15.5. Вкладка Advanced Compiler *диалогового окна* Project Options

После того как основная программа "попользовалась услугами" модуля DLL, она должна "убрать за собой" — выгрузить модуль из памяти. Эта операция выполняется функцией `FreeLibrary()`, которая входит в состав Win32 API. Функция принимает один аргумент — `HINSTANCE Dll`, который ранее был получен при вызове `LoadLibrary()`.

Этим исчерпываются изменения, которые нужно внести в проект для того, чтобы можно было использовать в нем функции из динамически загружаемого модуля DLL. Сохраните проект, скомпилируйте его и запустите приложение на выполнение.

В диалоговом окне работающего приложения щелкните на кнопке Dynamic — на экране появится окно с сообщением, которое "посылает" вам функция из динамически загруженного модуля DLL (рис. 15.6).

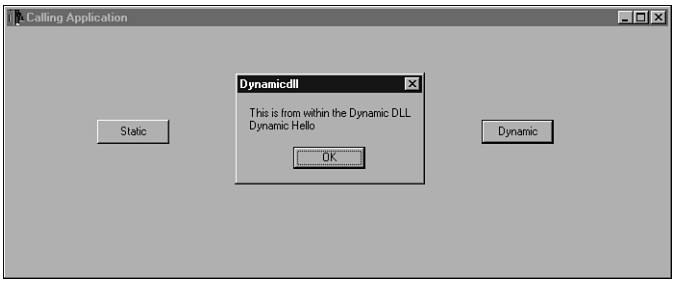


Рис. 15.6. Окно приложения `CallingApp`*, в котором выведено сообщение, сформированное в динамически связанном модуле DLL*

Вот и все, что я собирался рассказать вам о динамическом связывании DLL. Вполне возможно, что вам покажется, будто статическое связывание использовать легче. Это действительно так, но учтите, что статически связанный модуль остается в памяти все время, пока приложение выполняется, причем это не зависит от того, использует ли приложение размещенные в нем функции или нет.

Экспортирование классов

Теперь, после того как вы познакомились с методикой экспортирования функций, рассмотрим, а как же экспортируются целые классы.

Вновь откройте проект SimpleDll и добавьте в него новый модуль. Сохраните заготовку модуля под именем StaticClass.cpp. Откройте файл заголовка нового модуля — в него нужно будет внести определение класса, который мы собираемся экспортировать из DLL. Назовем новый класс TStaticClass. В нем будет один член-переменная, m_AlreadySaidHi, одно свойство, TimesCalled, и один метод, SayHi().

```
//
// Объявление класса TStaticClass
//
__declspec(dllexport) class TStaticClass
{
private:
    int FTimesCalled;
public:
    __fastcall TStaticClass ();
    __fastcall ~TStaticClass ();
     bool m_AlreadySaidHi;
     void __fastcall SayHi(void);
    __property int TimesCalled = { read = FTimesCalled} ;
} ;
```

Обращаю ваше внимание на то, что для этого класса используется практически та же методика экспортирования, что и для ранее рассмотренных функций. Единственное отличие — не используется квалификатор extern C. Класс “в экспортном исполнении ” объявляется почти так же, как обычный класс “для внутреннего пользования”.

Теперь откройте .cpp-файл реализации нового класса и добавьте в него программный код конструктора и деструктора. В конструкторе членам-переменным присваиваются значения по умолчанию.

```
__fastcall TStaticClass:: TStaticClass ()
{
    FTimesCalled    = 0;
    m_AlreadySaidHi = false;
}
//---------------------------------------------------------
__fastcall TStaticClass::~TStaticClass ()
{
}
```

После этого добавьте в этот же файл программный код реализации метода:

```
void __fastcall TStaticClass::SayHi(void)
{
    ShowMessage("Hello! From our DLL Class");
         // "Вас приветствует DLL-класс! "
    FTimesCalled++;
    m_AlreadySaidHi = true;
}
```

Введя новый код, скомпилируйте DLL.

Откройте проект CallingApp.bpr и в файл CallingApp.h добавьте следующие операторы:

```
#include "StaticClass.h"
...
private:
TStaticClass *StaticClass;
```

Откройте экранную форму проекта и добавьте в нее еще одну кнопку (объект компонента TButton). Установите в свойстве Caption нового объекта значение **Static Class**. Добавьте в файл реализации обработчик события OnClick новой кнопки:

```
void __fastcall TForm1::Button3Click(TObject *Sender)
{
    if (!StaticClass->m_AlreadySaidHi)
        StaticClass->SayHi();
    else
    {
        ShowMessage("You've already called this function " +
                  (AnsiString)StaticClass->TimesCalled +
                   " times.");
           // "Вы уже вызывали эту функцию "+ ... +" раз."
        StaticClass->SayHi();
    }
}
```

Теперь создадим объект StaticClass нового класса TStaticClass. Добавьте в конструктор класса экранной формы следующий программный код:

```
__fastcall TForm1::TForm1(TComponent* Owner)
    : TForm(Owner)
{
    StaticClass = new TStaticClass;
}
```

При закрытии диалогового окна динамически созданный объект должен быть удален, поэтому в деструктор класса экранной формы нужно добавить соответствующий оператор delete.

```
__fastcall TForm1::~TForm1()
{
    delete StaticClass;
}
```

Не забудьте добавить объявление деструктора и в файл заголовка:

```
public:
__fastcall TForm1::~TForm1();
```

Теперь сохраните и скомпилируйте проект CallingApp. Запустите его на выполнение и щелкните на кнопке Static Class в его диалоговом окне. Появится окно сообщения, в котором будет выведен текст Hello! From our DLL Class (рис. 15.7). Щелкните на этой кнопке еще несколько раз — после каждого очередного щелчка будет появляться еще одно окно сообщения, которое будет информировать вас, сколько раз вы уже выполнили эту нехитрую операцию.

Обращаю ваше внимание на то, что члены-переменные, свойства и методы класса, который экспортирует модуль DLL, используются в приложении точно так же, как если бы файлы этого класса были включены в состав основного проекта приложения.

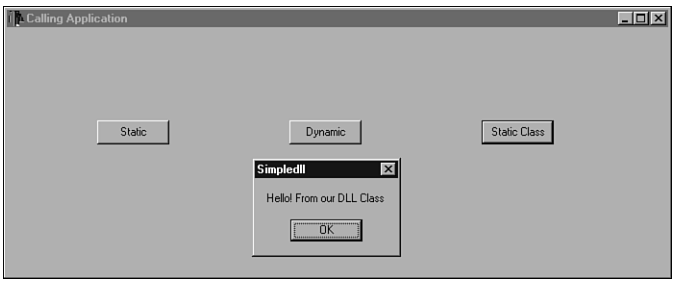


Рис. 15.7. Окно сообщения, сформированное объектом класса, программный код которого содержится в модуле DLL

Использование пакетов и модулей DLL

Пакеты, сформированные в среде C++Builder, представляют собой по существу те же модули DLL, в которые добавлены специфические расширения, необходимые для работы компонентов VCL. Использование пакетов обеспечивает лучшую совместимость с вызывающими приложениями, если они также созданы в среде C++Builder и используют VCL. Но существуют и некоторые нюансы, отличающие методику использования пакетов от методики использования DLL-модулей. Например, для загрузки и выгрузки пакетов нужно пользоваться функциями `LoadPackage()` и `FreePackage()` вместо `LoadLibrary()` и `FreeLibrary()`.

Если в пакете применяются такие же компоненты VCL, что и в вызывающем приложении, то они могут совместно использовать глобальную память. Таким образом, пакеты потенциально могут обладать б*о*льшей информацией о вызывающем приложении, чем модули DLL. Соответственно, и вызывающее приложение имеет б*о*льше возможностей следить за тем, что происходит в пакете. Основное преимущество использования пакетов проявляется в том случае, когда в пакете содержится дочерняя экранная форма многодокументного приложения (MDI-приложения). При создании дочерней экранной формы, размещенной в пакете, определить родительскую форму довольно просто. Если же дочерняя форма реализована в DLL-модуле, то для определения родительской формы приходится применять довольно хитроумные приемы.

Недостаток пакетов — возможность использования их в сочетании с вызывающими приложениями, созданными *только* в среде C++Builder. Если вы планируете использовать когда-либо динамически подгружаемый модуль совместно с приложениями, разработанными в другой среде, в частности в Visual C++, то единственный вариант — создавать его в виде DLL-модуля.

Познакомимся с методикой создания пакетов в среде C++Builder на практике. Выберите в системном меню команду File⇨New, а затем на вкладке New выберите Package (рис. 15.8). Далее создайте новый модуль — выберите в меню File⇨New, а на вкладке New выберите Unit. Сохраните файл пакета под именем `ThePackage.bpk`, а файл модуля — под именем `PackageFunctions.cpp`. Пока что все очень похоже на процесс создания DLL-модуля, который, я надеюсь, вы уже освоили.

Добавим в пакет все ту же простенькую функцию `PackageSay()`. Откройте файл заголовка и `PackageFunctions.h` и добавьте в него объявление функции:

```
extern "C" void __declspec(dllexport)PackageSay(char *WhatToSay);
```

В файл реализации `dllimport.cpp` добавьте текст функции:

```
void PackageSay(char *WhatToSay)
{
    ShowMessage(WhatToSay);
}
```

Дайте команду на компиляцию пакета, и C++Builder сформирует файл с расширением `.bpl`. Это и есть загрузочный файл пакета. По умолчанию C++Builder размещает его в подкаталоге `Projects\Bpl` основного каталога `C++Builder`. Если вас это не устраивает, настройку среды можно изменить. Откройте диалоговое окно Project Options (для этого нужно выбрать в меню команду Project⇨Options), а в нем — вкладку Directories/Conditionals. Самое нижнее поле редактирования в зоне Directories на этой вкладке называется BPI/LIB Output. В нем-то и нужно указать, в каком каталоге вы хотите размещать создаваемые файла пакетов (рис. 15.9).

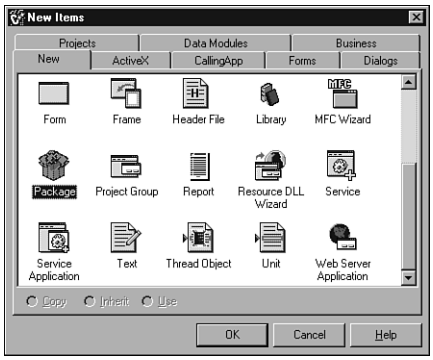


Рис. 15.8. Диалоговое окно New Items

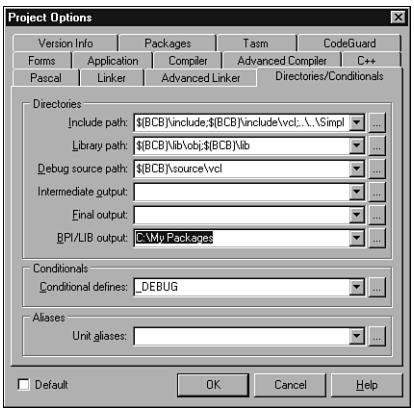


Рис. 15.9. Вкладка Directories/Conditionals *диалогового окна* Project Options

Скопируйте созданный `.bpl`-файл в ту же папку, в которой находятся файлы приложения `CallingApp`.

Теперь вновь откройте проект `CallingApp`. Добавьте в его экранную форму еще одну кнопку (объект компонента `TButton`). Установите в свойстве `Caption` нового объекта значение **`Package`**. Добавьте в файл реализации обработчик события `OnClick` новой кнопки, текст которого приведен в листинге 15.2.

Листинг 15.2. Обработчик события `Button4Click`

```
void __fastcall TForm1::Button4Click(TObject *Sender)
{
    // Сначала нужно загрузить пакет.
    Windows::HINST Package = LoadPackage("ThePackage.bpl");

    // Проверим, загружен ли пакет.
    if (Package)
    {
        // Извлечение адреса функции.
        SayFromPackage = (SayType *)GetProcAddress(
                  (HINSTANCE)Package, "_PackageSay");
        // Если адрес получен, вызвать функцию.
        if (SayFromPackage)
            SayFromPackage("Package Hello");
```

```
        else
            ShowMessage(SysErrorMessage(GetLastError()));

        // Выгрузить пакет.
        UnloadPackage(Package);
    }
    else
    {
        ShowMessage(SysErrorMessage(GetLastError()));
        ShowMessage("Unable to load the Package");
                  // "Пакет не загружен!";
    }
}
```

В файл заголовка проекта добавьте объявление нового члена:

```
private:
SayType *SayFromPackage;
```

Обратите внимание на то, что в объявлении используется тот же тип, который мы создали ранее для работы с динамически загружаемым модулем DLL.

Единственное, что в этом программном коде связано со спецификой пакетов, — вызов функций загрузки и выгрузки `LoadPackage()` и `UnloadPackage()`. Все остальное выполняется так же, как и при использовании динамически загружаемых модулей DLL. Для получения адреса функции используется все та же API-функция `GetProcAddress()`.

Уже в который раз сохраните этот проект и скомпилируйте его. Затем запустите приложение `CallingApp` на выполнение и щелкните на кнопке Package в его диалоговом окне — появится окно сообщения, показанное на рис. 15.10.

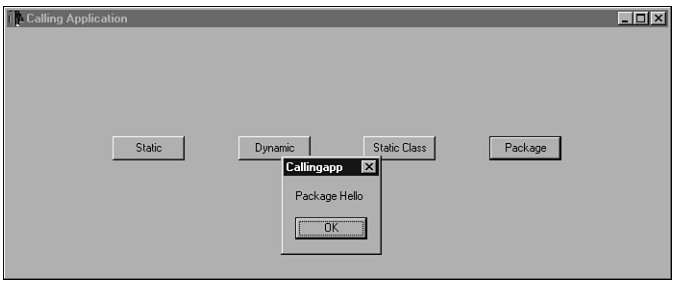


Рис. 15.10. В окне приложения `CallingApp` выводится сообщение, сформированное пакетом

Экранные формы однодокументных приложений в DLL

Теперь рассмотрим, как поместить в модуль DLL средства графического интерфейса с пользователем — экранную форму. Для этого используется та же методика, которую мы применяли при включении функций в состав модуля DLL.

Мы не будем приводить здесь весь программный код, который необходим для включения класса экранной формы в модуль DLL, а только самые существенные фрагменты. Они взяты из проекта `MDIChildDLL.bpr`, файлы которого вы найдете в папке `source\Chapter15\MDIDLL` на прилагаемом к книге компакт-диске.

1. Добавьте в DLL экранную форму.
2. Объявите экспортируемой функцию `ShowMyForm()`, как показано ниже:
   ```
   extern "C" void __declspec(dllexport)ShowMyForm(void);
   ```
3. Добавьте текст этой функции в `.cpp`-файл.
4. Сформируйте объект экранной формы и выведите его на экран:
   ```
   TForm *MyForm = new Tform(NULL);
   MyForm->ShowModal();
   delete MyForm;
   ```

В принципе, это все, что требуется от разработчика для того, чтобы в DLL-модуле имелась своя экранная форма. Если вы хотите посмотреть, как это будет работать в приложении, поместите приведенный код в динамически загружаемый модуль DLL — после запуска приложения вы увидите картинку, показанную на рис. 15.11.

Рис. 15.11. В родительском окне MDI-приложения выведена экранная форма DLL-модуля

Дочерние экранные формы MDI в модулях DLL

Когда речь заходит о применении модулей DLL, очень часто можно услышать вопрос “Как поместить в DLL дочернюю экранную форму MDI?” Суть проблемы в том, что объект класса дочернего окна не может работать, если ему “не известно” родительское окно. Приходится применять разные хитроумные способы, чтобы “обмануть” DLL и представить этот модуль как часть приложения, которое и включает в себя родительское окно.

Создайте с помощью мастера ***DLL Wizard*** модуль DLL и сохраните его файлы: файл проекта под именем MDIChildDLL.bpr, а файл реализации — под именем MDIChildDLL.cpp. Включите в проект экранную форму и установите значения ее свойств:

- Name = Child
- FormStyle = fsMDIChild
- Caption = MDI Child
- Height = 200
- Width = 400

Создайте обработчик события OnClose для класса экранной формы и включите в него оператор

```
Action = caFree;
```

Поместите на поле формы надпись I'm a MDI Child in a DLL!. Сохраните файл экранной формы под именем MDIChild.cpp.

Нам понадобится также экспортируемый метод, который можно будет использовать для вызова этой экранной формы. Откройте файл реализации главного модуля проекта MDIChildDLL.cpp и добавьте в него следующий программный код:

```
#include "MDIChild.h"
TApplication *ThisApp = NULL;
```

Объявление функции можно включить прямо в этот файл, причем оно будет предшествовать тексту программы самой функции.

```
extern "C" void
    __declspec(dllexport)ShowMDIChildForm(
                        TApplication *CallingApp);

void ShowMDIChildForm(TApplication *CallingApp)
{
    if (!ThisApp)
    {
        ThisApp = Application;
        Application = CallingApp;
    }
    Child = new TChild(Application);
    Child->Show();
}
```

Функция ShowMDIChildForm имеет только один аргумент, причем его тип — TApplication. Это сделано для того, чтобы через аргумент передать в модуль указатель на объект вызывающего приложения и присвоить этот указатель члену ThisApp. В результате дочернее окно получит окно-родителя.

Теперь нужно позаботиться и об обратном преобразовании объекта Application. Для этого воспользуемся функцией DllEntryPoint(). Эта функция должна принять такой вид:

```
int WINAPI DllEntryPoint(HINSTANCE hinst, unsigned long reason,
                         void* lpReserved)
{
    // Если DLL выгружается, вернуть
    // объекту Application исходное значение.
```

```
    if ( (reason == DLL_PROCESS_DETACH) && (ThisApp) )
        Application = ThisApp;
    return 1;
}
```

Функция `DllEntryPoint()` имеет аргумент `reason`, который несет информацию о предстоящих действиях вызывающего процесса. Ниже перечислены возможные значения этого аргумента:

- `DLL_PROCESS_ATTACH`
- `DLL_THREAD_ATTACH`
- `DLL_THREAD_DETACH`
- `DLL_PROCESS_DETACH`

Сейчас нас интересует только значение `DLL_PROCESS_DETACH`. Если `reason` имеет именно это значение, нужно восстановить объект приложения. (Более подробную информацию по этой теме вы можете получить в документе *WIN32 SDK Reference*.)

Прежде чем сохранить файлы проекта и компилировать модуль DLL, нужно выполнить еще одну операцию. Дело в том, что операции с объектом приложения в DLL, о которых шла речь, позволяют DLL "видеть" родительскую MDI-форму, но при этом сама родительская форма "не знает" о существовании дочерних MDI-форм. Чтобы объект родительской формы "знал" о существовании дочерних MDI-форм, нужно компилировать модуль DLL вместе с динамической библиотекой RTL и пакетом Runtime, а для этого потребуется внести изменения в настройку проекта. Откройте диалоговое окно Project Options (выберите в меню команду Project⇨Options), а затем — его вкладку Packages. В самом низу этой вкладки установите флажок Build with Runtime Packages (рис. 15.12). Затем откройте вкладку Linker и проверьте, установлен ли в ней флажок Use Dynamic RTL (рис. 15.13). Теперь можно сохранить проект и компилировать его.

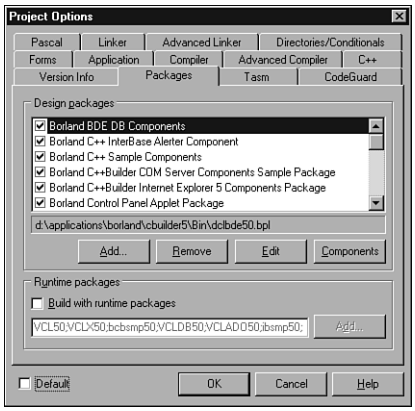


Рис. 15.12. Вкладка Packages диалогового окна Project Options

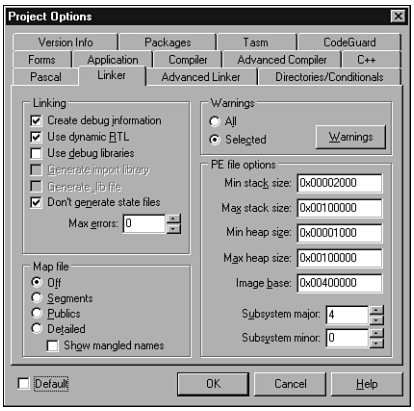


Рис. 15.13. Вкладка Linker диалогового окна Project Options

Следующий этап — создание вызывающего приложения, в котором будет находиться объект родительской MDI-формы.

Создайте новый проект и установите следующие свойства объекта главной экранной формы приложения:

- `Name = Parent`
- `Caption = MDI Parent Form`
- `FormStyle = fsMDIForm`

Добавьте в экранную форму компонент TMainMenu, причем в его свойстве Caption установите значение **&File**. Затем добавьте в раздел File этого меню два пункта. В свойстве Caption первого пункта установите значение **&New**, а в таком же свойстве второго — значение **E&xit**.

Сохраните файл реализации под именем ParentForm.cpp, а файл проекта — под именем MDIParent.bpr. Включите в файл реализации подпрограмму обработки события OnClick для элемента меню &New, текст которой приведен в листинге 15.3.

Листинг 15.3. Обработчик события New1Click()

```
void __fastcall TParent::New1Click(TObject *Sender)
{
    typedef void
      __declspec(dllimport)ShowChildType(TApplication *);
    ShowChildType *ShowMDIChild;

     // Проверить, загружен ли модуль DLL.
    if (!Dll)
        Dll = LoadLibrary("MDIChildDLL.dll");
    if (Dll)
    {
        // Получить адрес функции.
        ShowMDIChild = (ShowChildType *)GetProcAddress(Dll,
                    "_ShowMDIChildForm");
        // Если адрес функции получен, вызвать ее.
        if (ShowMDIChild)
            ShowMDIChild(Application);
        else
        {
            ShowMessage(SysErrorMessage(GetLastError()));
            // Если не удалось получить адрес функции,
            // выгрузить модуль DLL из памяти.
            FreeLibrary(Dll);
        }

    }
    else
    {
        ShowMessage(SysErrorMessage(GetLastError()));
        ShowMessage("Unable to load the DLL");
                  // "Модуль DLL не загружен!"
    }
}
```

Подобный текст уже встречался в прежних упражнениях, не так ли? Практически такой же программный код использовался в проекте DynamicDLL, с той лишь разницей, что на сей раз в самой подпрограмме объявление HINSTANCE Dll отсутствует — оно перенесено в файл заголовка, а процедура загрузки DLL выполняется только один раз — при первом обращении в обработчику. Если в начале этой программы не будет проверяться, загружен ли уже модуль DLL, а сразу будет вызываться функция LoadLibrary(), то это приведет к наращиванию значения в счетчике использования DLL. При этом повторная загрузка выполняться не будет.

Такой прием может оказаться полезным в том случае, когда в приложении имеется множество функций, требующих обращения к модулю DLL. Но при этом каждому вызову `LoadLibrary()` должен соответствовать и вызов функции `FreeLibrary()`. Если функция `LoadLibrary()` будет вызвана дважды, а `FreeLibrary()` только один раз, в счетчике ссылок останется единица и модуль DLL не будет выгружен до тех пор, пока еще где-либо в программе не встретится вызов `FreeLibrary()`.

В файл заголовка добавьте строку:

```
private:
    HINSTANCE Dll;
```

Опять откройте `.cpp`-файл и вставьте в него конструктор:

```
__fastcall TParent::TParent(TComponent* Owner)
    : TForm(Owner)
{
    Dll = NULL;
}
```

Остается последняя операция — обеспечить перед закрытием главного окна проверку удаления объектов всех дочерних MDI-форм. Добавьте обработчик события `OnClose` родительской MDI-формы:

```
void __fastcall TParent::FormClose(TObject *Sender, TCloseAction &Action)
{
    // Проверка, удалены ли объекты всех дочерних MDI-форм.
    while (MDIChildCount)
        MDIChildren[0]->Free();

    FreeLibrary(Dll);
}
```

В этой программе организуется цикл `while(MDIChildCount)` по открытым в данный момент объектам `MDIChidren`. Для каждого из них выполняется оператор `MDIChidren[0]->Free()`, причем после удаления каждого очередного объекта дочерней MDI-формы в списке происходит перегруппировка, и указатель на очередной объект опять имеет индекс `0`.

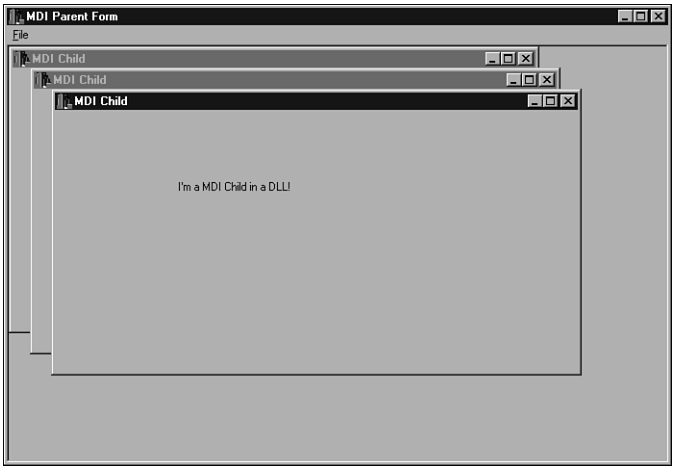


Рис. 15.14. Главное окно MDI-приложения и три открытых дочерних окна

Если объекты дочерних MDI-форм содержатся в том же выполняемом файле, что и объект родительской формы, об удалении можно не заботиться — это происходит автоматически.

При компилировании этого приложения также должны быть установлены опции использования динамической библиотеки RTL и пакета Runtime. Как выполняется эта настройка среды C++Builder, было рассказано чуть выше.

Сохраните файлы проекта, скомпилируйте приложение и запустите его на выполнение. После каждого вызова команды меню File⇨New на экране будет появляться новое дочернее окно (рис. 15.14). Каждое из них можно закрывать индивидуально в любом порядке. Все они закроются автоматически при завершении работы приложения.

Дочерние экранные формы MDI в пакетах

Использование дочерних MDI-форм в составе пакета имеет определенные преимущества перед их использованием в составе обычных модулей DLL. Хотя в принципе процесс идет так же, приложение, в котором применяется пакет, работает более устойчиво. Поскольку при такой организации работы программы вызывающее приложение и пакет используют одни и те же глобальные ресурсы, объект дочерней MDI-формы всегда "видит" объект родительской формы, и наоборот.

Чтобы продемонстрировать такую организацию программы, мы воспользуемся созданным ранее проектом вызывающего приложения `MDIChildDll`, но создадим новый пакет. При желании вы можете скопировать файл этого пакета `MDIChildPackage.bpk` из папки `source\Chapter15\MDIPackage` на прилагаемом к книге компакт-диске.

Выберите в меню команду File⇨New и откройте вкладку Package (с этой процедурой вы уже знакомы по одному из прежних упражнений). Добавьте в пакет экранную форму и сохраните ее файл под именем `MDIChildPkForm.cpp`, а пакет — под именем `MDIChildPackage.bpk`.

Установите свойства объекта экранной формы такими же, как и при создании дочерней формы в DLL:

- `Name = Child`
- `FormStyle = fsMDIChild`
- `Caption = MDI Package Child`
- `Height = 200`
- `Width = 400`

Создайте обработчик события `OnClose` объекта формы и включите в него оператор `Action = caFree;`. Включите в файл реализации `MDIChildPackage.cpp` директиву:

```
#include "MDIChildPkForm.h"
```

В файл `MDIChildPackage.cpp` вставьте приведенный ниже программный код:

```
extern "C" void __declspec(dllexport)ShowMDIChildPkForm(void);

void ShowMDIChildPkForm(void)
{
    Child = new TChild(Application);
    Child->Show();
}
```

Эта функция очень напоминает рассмотренную в предыдущем примере экспортируемую функцию DLL, но не имеет аргументов и не выполняет переключение объекта приложения. Нам также не понадобился файл заголовка, поскольку пакет не планируется статически связывать с вызывающим приложением. Но если статическое связывание все-таки планируется, нужно сформировать файл заголовка, поместить в него прототип функции, а в файл `MDIChildPackage.cpp` включить соответствующую директиву `#include`.

Теперь можно сохранить файлы и выполнить компиляцию пакета. Не забудьте, что C++Builder поместит скомпилированный файл в каталог `Projects\Bpl` и придется скопировать его в тот каталог, где находится приложение `MDIParent`.

Откройте существующий проект `MDIParent` и добавьте в меню (компонент `TMainMenu`) новый пункт. Разместите его вслед за пунктом `&New` и установите в свойстве `Caption` нового элемента значение **`New &Package`**. Создайте для этого элемента обработчик события `OnClick` и вставьте в него программный код, приведенный в листинг 15.4.

Листинг 15.4. Обработчик события `NewPackage1Click`

```
void __fastcall TParent::NewPackage1Click(TObject *Sender)
{
    typedef void __declspec(dllimport)ShowChildPkType(void);
    ShowChildPkType *ShowPackageChild;
     // Проверить, загружен ли пакет.
    if (!Package)
        Package = LoadPackage("MDIChildPackage.bpl");
    if (Package)
    {
        // Получить адрес функции.
        ShowPackageChild =
            (ShowChildPkType *)GetProcAddress((HINSTANCE)Package,
                   "_ShowMDIChildPkForm");
        // Если адрес функции получен, вызвать ее.
        if (ShowPackageChild)
            ShowPackageChild();
        else
        {
            ShowMessage(SysErrorMessage(GetLastError()));
            // Если не удалось получить адрес функции,
            // выгрузить пакет из памяти.
            UnloadPackage(Package);
        }

    }
    else
    {
        ShowMessage(SysErrorMessage(GetLastError()));
        ShowMessage("Unable to load the Package");
                  // "Пакет не загружен!"
    }
}
```

Этот код очень похож на приведенный в листинге 15.3 код обработчика `New1Click()`, который позволял обращаться к модулю DLL. Единственное отличие — вместо загрузки DLL производится загрузка пакета (вызывается функция `LoadPackage()`) и при вызове функции из пакета ей не передаются аргументы.

Нам осталось модифицировать в этом приложении обработчик события `OnClose` объекта родительской экранной формы. Новый вариант обработчика должен выглядеть, как представлено ниже:

```
void __fastcall TParent::FormClose(TObject *Sender,
      TCloseAction &Action)
{
    // Проверка, удалены ли объекты всех дочерних MDI-форм.
    while (MDIChildCount)
        MDIChildren[0]->Free();

    if (Dll)
        FreeLibrary(Dll);
    if (Package)
        UnloadPackage(Package);
}
```

В новом варианте также используется цикл `while` для закрытия всех дочерних MDI-окон, но добавлены операторы удаления модуля DLL и выгрузки пакета. Для выполнения последней операции вызывается функция `UnloadPackage()`.

Сохраните файлы приложения `MDIParent`, скомпилируйте его и запустите на выполнение. В меню File нового приложения для создания дочерних окон можно вызывать две команды: New и New Package. При вызове команды New используется обращение к DLL и формируется дочернее окно MDI Child. При вызове команды New Package используется обращение к пакету и формируется дочернее окно MDI Package Child. На рис. 15.15 показано окно приложения `MDIParent`, в котором имеются дочерние окна обоих видов.

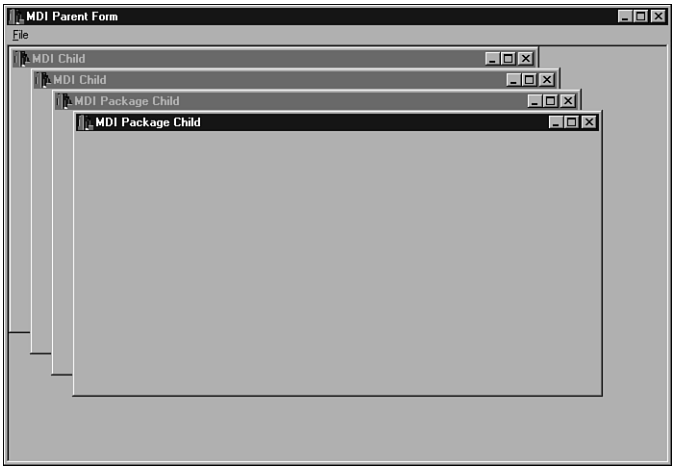


Рис. 15.15. Главное окно MDI-приожения, в котором открыты дочерние окна, созданные с помощью DLL и пакета

Использование модулей DLL, созданных в среде Microsoft Visual C++

Иногда в проекте, создаваемом в среде C++Builder, нужно обратиться к функциям, которые находятся в модуле DLL, созданном в среде Visual C++. Это осуществимо, но при создании приложения следует выполнить определенные подготовительные операции. Если планируется использовать динамическое связывание, то это выполняется точно так же, как и при работе с DLL, созданными в C++Builder.

А вот если планируется использовать статическое связывание, то придется сформировать `.lib`-файл, который C++Builder смог бы воспринять и с которым смог бы связать приложение. Причина в том, что в среде Visual C++ используется несколько отличный механизм экспортирования функций. По-иному выглядят и имена функций, поэтому в `.lib`-файле должны соблюдаться соглашения об именовании экспортируемых функций, которые приемлемы для компоновщика C++Builder.

Для создания `.lib`-файла, соответствующего DLL из Visual C++, нужно воспользоваться инструментальной программой `Coff2Omf.exe`, которая запускается из командной строки. Эта программа находится в папке `Bin` корневого каталога C++Builder. Программа `Coff2Omf.exe` при запуске принимает два аргумента. Один из них — имя файла исходной библиотеки, другой — имя создаваемого файла. Например:

```
Coff2Omf MyDll.lib MyDll.new
```

После того как новый файл создан, можно удалить исходный файл, а новый переименовать, дав ему расширение `.lib`. Другой вариант: перенесите новую библиотеку в другой каталог, а потом измените расширение. В результате этой процедуры будет создан новый файл импортируемой библиотеки, который нужно включить в состав проекта C++Builder. Используя этот измененный файл, среда C++Builder сможет корректно выполнить компоновку.

Если эта методика не даст ожидаемого результата, выясните, как организовано экспортирование функций в DLL, и присвойте функциям псевдонимы, воспринимаемые C++Builder. Для этого сначала с помощью утилиты `Impdef.exe` создайте файл определений (он имеет расширение `.def`), который позволит посмотреть имена всех экспортируемых DLL функций и их порядковые номера. Далее измените экспортируемые функции в `.def`-файле так, чтобы функции в нем имели такой вид:

```
Старый раздел экспорта
EXPORTS
    _Add@8                          =_Add               @1

Новый раздел экспорта
EXPORTS
    Add=_Add@8
```

Сохраните отредактированный `.def`-файл и воспользуйтесь утилитой `Implib.exe` для того, чтобы создать на основе этого файла новый файл библиотеки. Этот файл должен "понравиться" C++Builder. Утилита `Implib.exe` также принимает два параметра: имя файла создаваемой библиотеки и имя `.def`-файла, который служит источником информации для библиотеки. Например:

```
Implib MyDll.lib MyDll.def
```

Поскольку теперь в вашем распоряжении имеется библиотека в формате, "полюбившемся" C++Builder, можно включить ее в проект.

В папке source\Chapter15\VCppProject на прилагаемом к книге компакт-диске имеется комплект файлов C++Builder-проекта CallVCppDll.bpr, в котором используется модуль DLL, созданный в среде Visual C++.

Использование DLL, созданных в C++Builder, совместно с приложением Microsoft Visual C++

Предположим теперь, что вы создали модуль DLL и присвоили ему уникальный номер из *n* цифр. Основное назначение модуля — работа с C++Builder-приложениями, но требуется чтобы этот модуль работал и совместно с приложением, созданным в среде Visual C++. Это можно сделать без особых хлопот, если заранее известить мастер ***DLL Wizard*** среды C++Builder, что модуль должен работать "в связке" не только с "родными" приложениями, но и с приложениями Visual C++. Для этого в окне настройки мастера нужно установить флажок Visual C++ style DLL.

При динамическом связывании вообще не должно быть никаких проблем с использованием DLL, а при статическом связывании проблемы могут появиться. Причина все в тех же отличиях форматов экспортирования, используемых в средах C++Builder и Visual C++.

Для динамического связывания с DLL, созданной в среде C++Builder DLL, в приложении Visual C++ должна использоваться методика, описанная ранее применительно к C++Builder.

- Вызовите функцию LoadLibrary() для загрузки в память модуля DLL.
- С помощью функции GetProcAddress() получите адрес экспортируемой функции.
- После завершения работы с модулем DLL удалите его из памяти с помощью функции FreeLibrary().

Это все, что нужно сделать для того, чтобы DLL, созданная в C++Builder, выполняла в режиме динамического связывания "заявки" приложения, созданного в среде Visual C++.

При статическом связывании возникает проблема, вызванная использованием в средах Visual C++ и C++Builder двух разных форматов: OMF и COFF. Для того чтобы справиться с этой проблемой, нужно воспользоваться утилитой IMPDEF.EXE, которая находится в папке C++Builder\Bin, и сформировать с ее помощью .def-файл. Затем в этот файл нужно внести изменения — удалить символы подчеркивания в начале имен экспортируемых функций.

В каталоге source\Chapter15\VCppProject на прилагаемом компакт-диске находится комплект файлов проекта VCppDLL.bpr, созданного в среде C++Builder, и комплект файлов проекта BCBCallingApp.dsw, созданного в среде Visual C++. Приложение BCBCallingApp должно использовать DLL из проекта VCppDLL. В папке VCppProject содержатся выполняемые файлы приложения Visual C++ и DLL C++Builder.

Запустите на выполнение программу Impdef с помощью следующей команды:

```
Impdef VCppDLL.def VCppDLL.dll
```

В созданном .def-файле вы увидите следующий текст:

```
LIBRARY     VCPPDLL.DLL

EXPORTS
    _GetNumber                  @1   ; _GetNumber
    ___CPPdebugHook             @2   ; ___CPPdebugHook
```

Удалите символ подчеркивания в имени функции `_GetNumber` и сохраните отредактированный файл.

После этого нужно сформировать `.lib`-файл, который сможет распознать приложение Visual C++. Этот файл создается программой `Lib.exe`, которая входит в комплект Visual C++ и находится в каталоге `VisualC++\Bin`.

Программа запускается на выполнение следующей командой:

```
LIB /DEF:VCppDLL.def
```

В результате будет создан файл `VCppDLL.lib`, который нужно включить в состав проекта Visual C++.

Разработка подключаемых компонентов

Далее мы рассмотрим специальный тип модулей DLL, которые в практике программирования получили наименование *подключаемые компоненты* (*plug-in*). Подключаемые компоненты позволяют элегантно решить некоторые проблемы, связанные с использованием DLL.

Наиболее серьезной проблемой является интерфейс между приложением и модулем DLL. Разработчик приложения должен знать формат обращения к функциям и данным, размещенным в DLL. Доступ к интерфейсу открывается либо через файл заголовка для библиотеки, либо, в случае динамического или позднего связывания, через функции Windows API `LoadLibrary()`, `GetProcAddress()` и `FreeLibrary()`.

В любом случае при разработке приложения нужно точно знать формат обращения к функциям, размещенным в DLL. Такой подход затрудняет сопровождение программного обеспечения:

- выявление и устранение ошибок в программе или модификация программы требует полной повторной компиляции и рассылки пользователям файлов большого размера;
- приложения фактически не являются масштабируемыми; расширение функциональных возможностей приложения требует почти полной его переработки.

Структура подключаемых компонентов

С помощью подключаемых компонентов можно расширить функциональные возможности приложения в процессе эксплуатации. Предположим, вы хотите предоставить потенциальным пользователем разработанной вами программы возможность опробовать. Пробная версия программы, как правило, имеет неполный спектр функциональных возможностей и ограниченное время эксплуатации. Все это довольно просто организовать, если при проектировании программы ориентироваться на использование подключаемых компонентов. Это означает, что в приложение включаются *интерфейсы* с подключаемыми компонентами, а собственно функциональные возможности программы “закладываются” в подключаемые компоненты. В пробную версию включается один набор подключаемых компонентов (Trial Version plug-in), а в продаваемую — другой (Full Version plug-in), причем в этом втором реализуется полный спектр функций.

Еще одно преимущество подключаемых компонентов — меньший размер, по сравнению с полной программой, а это весьма существенный фактор при распространении таких компонентов по сети Internet.

Базовая структура подключаемых компонентов

Поскольку подключаемые компоненты — это один из видов модулей DLL или пакетов, то методы доступа к ним очень похожи на те, которые мы рассматривали выше. Основное отличие — способ определения интерфейса между вызывающим приложением и библиотекой функций компонента. В "обычном" модуле DLL для этого используются функции `LoadLibrary()`, `GetProcAddress()` и `FreeLibrary()`, которые входят в состав Windows API. В листинге 15.5 показано, как это делается (листинг включен в этот раздел просто чтобы напомнить вам изложенное в предыдущем разделе).

Листинг 15.5. Обычная процедура доступа к функциям, размещенным в модуле DLL

```
typedef double (__stdcall *ADDFUNCTIONPTR)(double,double);
HINSTANCE dllInstance;
ADDFUNTIONPTR dllAddFunctionAddress;
DllInstance = ::LoadLibrary("MyDll.dll");
DllAddFunctionAddress = (ADDFUNCTIONPTR)
             ::GetProcAddress(dllInstance, "Add");
FreeLibrary (dllInstnce);
```

В этом фрагменте загружается модуль DLL `MyDLL.dll` и извлекается указатель на функцию `Add()`, которая принимает два аргумента типа `double` и возвращает значение типа `double`.

В первой строке программы объявляется тип указателя на функцию. В дальнейшем переменная этого типа будет использована для хранения указателя на определенную функцию в DLL. Обращаю ваше внимание на то, что с функцией можно выполнять только две операции — вызывать ее или получить адрес функции. Формат указателя на функцию следующий:

```
<тип возвращаемого значения>(*<имя указателя на функцию>)(
↳arg1, arg2, ... , argn);
```

Работать с таким длинным выражением при каждом объявлении нового указателя довольно неудобно, поэтому мы объявили новый тип `ADDFUNCTIONPTR` такого указателя. Двумя строками ниже объявляется новая переменна `dllAddFunctionAddress` этого типа. Далее выполняется загрузка модуля DLL и формируется дескриптор этого модуля `DllInstance`. После загрузки модуля (фактически, библиотеки функций) вызывается функция Windows API `GetProcAddress()`, которой в качестве аргументов передаются дескриптор модуля DLL и имя той функции из библиотеки, указатель на которую требуется получить.

В первой строке листинга 15.5 вы видите объявление `__stdcall`. Если это объявление опустить, то придется добавлять символ подчеркивания к имени каждой функции, которую экспортирует модуль DLL. Отыскав заданную функцию в библиотеке, `GetProcAddress()` возвращает ее адрес, который сохраняется в переменной `dllAddFunctionAddress`. Поскольку этот фрагмент мы представили только для того, чтобы продемонстрировать обычную методику получения указателя на функцию, то после получения указателя модуль DLL удаляется из памяти.

Теперь рассмотрим процесс разработки простого подключаемого компонента. Такой простой компонент будет содержать несколько экспортируемых функций. В данном случае это будет функция построения простой геометрической фигуры — треугольника. Мы создадим еще один аналогичный по структуре компонент с функциями построения окружности. Естественно, что такие геометрические фигуры можно рассматривать как классы-наследники базового класса, представляющего обобщенную геометрическую фигуру. В

листинге 15.6 представлено определение базового класса BaseShape, в листинге 15.7 — определение производного класса TriangleShape, а в листинге 15.8 — определение производного класса CircleShape.

Листинг 15.6. Определение базового класса BaseShape

```
class BaseShape
{
public:
    BaseShape(){ } ;
    ~BaseShape(){ }
    virtual char* GetShapeType() = 0;
} ;
```

Листинг 15.7. Определение класса TriangleShape

```
class TriangleShape :  public BaseShape
{
public:
    TriangleShape() : BaseShape() { } ;
    ~TriangleShape();
    virtual char* GetShapeType();
} ;
TriangleShape::GetShapeType()
{
    return "Triangle";
}
```

Листинг 15.8. Определение класса CircleShape

```
class CircleShape :  public BaseShape
{
public:
    CircleShape() : BaseShape(){ } ;
    ~CircleShape();
    vritual char* GetShapeType();
} ;
CircleShape::GetShapeType()
{
   return "Circle";
}
```

В программе мы могли бы просто создать объект любого класса и вызывать методы GetShapeType() этих объектов. При работе с подключаемыми компонентами это делается по-другому. Все подключаемые компоненты разрабатываются под заранее заданный интерфейс. Другими словами, разработчик вызывающего приложения извещает всех разработчиков подключаемых компонентов, что в приложении будет организован поиск в DLL функции с определенным именем или функции, возвращающей результат определенного типа. В данном случае приложение будет отыскивать функцию, которая не принимает никаких аргументов и возвращает указатель на тип char. Определение базового класса будет известно вызывающе-

му приложению через заголовочный файл. Поскольку в структуре базового класса имеется виртуальный метод, в производных классах этот метод должен быть перегружен. Вызывающему приложению не известно, какой будет фигура, вызванная с помощью этого метода, поэтому нужно изыскать способ, позволяющий вызывать все типы фигур. Эта идея реализуется классом GenericShape, определение которого приведено ниже.

```
class GenericShape : public BaseShape
{
    GenericShape() : BaseShape(){ } ;
    ~GenericShape(){ } ;
    virtual char* GetShapeName();
}
```

Для вызова подключаемого компонента в реализацию метода GetShapeName() будет включено обращение к функции GetProcAddress().

В модули DLL подключаемых компонентов нужно включить всего одну функцию, помимо программного кода, что автоматически создается мастером DLL Wizard. Ниже приведен ее текст для компонента, который формирует окружность:

```
extern "C" char* __declspec(dllexport) __stdcall GetShapeName()
{
    // В компоненте, который формирует треугольник, функция
    // с таким же именем должна будет вернуть слово "triangle".
    return "Circle";
}
```

Итак в DLL подключаемого компонента имеется экспортируемая функция, которая возвращает значение типа char*.

В вызывающем приложении загрузка DLL и вызов экспортируемой функции GetShapeName() осуществляется в методе GetShapeName() класса GenericShape, текст которого представлен в листинге 15.9.

Листинг 15.9. Метод GetShapeName() класса GenericShape

```
char* GenericShape::GetShapeName()
{
    typedef char* (__stdcall *SHAPENAMEADDR)();
    HISTANCE plugin;
    SHAPENAMEADDR shapeNameFunc;
    plugin = ::LoadLibrary("ShapePlugin.dll");
    if (plugin)
    {
        shapeNameFunc =
           (SHAPENAMEADDR)GetProcAddress(plugin,"GetShapeName");
        if (shapeNameFunc)
            char* tempName = shapeNameFunc();
        FreeLibrary(plugin);
        return tempName;
    }
    ShowMessage("Error! Couldn't load library");
               // "Ошибка! Библиотека не загружается."
    return 0;
}
```

В приведенном тексте программы загружается подключаемый компонент, который существует в виде DLL-модуля ShapePlugin. Если такой DLL-модуль не удается найти на компьютере, формируется сообщение об ошибке и возвращается значение 0. В противном случае в загруженной библиотеке отыскивается GetShapeName(), которая не имеет аргументов и возвращает значение типа char*. Если такая функция обнаруживается в библиотеке, ее адрес фиксируется в переменной shapeNameFunc, после чего функция вызывается на выполнение и возвращает наименование геометрического объекта, который содержится в подключенном компоненте. После этого модуль DLL выгружается из памяти, а полученное от компонента сообщение возвращается в вызывающую программу.

Такой стиль использования подключаемых компонентов предполагает, что весь интерфейс работы с компонентом (соответствующий программный код) хранится в вызывающем приложении, а в самом компоненте хранится только код реализации определенных функций. Имея набор компонентов с разными вариантами реализации одних и тех же функций, можно формировать версии приложения с разными функциональными возможностями.

Но в такой методике работы с подключаемыми компонентами есть один недостаток — она не очень надежна. В следующем разделе мы рассмотрим другую, более современную методику, основанную на модели составных объектов COM. Эта методика позволяет изолировать разработчика от программирования низкоуровневых функций доступа к функциям DLL.

Класс TIBCB5PlugInBase

В составе среды C++Builder имеются все необходимые инструментальные средства для создания подключаемых компонентов и приложений, работающих с такими компонентами. Загляните в файлы исходных текстов программ из набора ***Open Tools API***. Классы в ***Open Tools API*** инкапсулируют все методы, необходимые для работы с подключаемыми компонентами.

Все классы в составе Open Tools API являются производными от класса TInterface. Нас будет интересовать один из его "наследников" — класс TIExpert. Как и у всех других классов библиотеки VCL, его имя начинается с буквы T, которая означает *type* (тип). Второй буквой в именах классов из состава ***Open Tools*** чаще всего бывает I, которая означает *interface* (интерфейс). Класс интерфейса определяет, как будут работать все производные от него классы. В языке C++ интерфейс — это класс, который располагает только объявлениями методов без их реализации. В тексте объявления класса все его методы имеют квалификатор virtual, а завершается строка объявления метода приравниванием к 0 (= 0). Нельзя создать объект такого класса, но можно создавать объекты производных от него классов, в которых имеется реализация виртуальных методов. Взгляните на объявление класса TIExpert в листинге 15.10.

Листинг 15.10. Объявление класса TIExpert в среде Delphi

```
TIExpert = class(TInterface)
  public
    {  Expert UI strings }
    function GetName: string; virtual; stdcall; abstract;
    function GetAuthor: string; virtual; stdcall; abstract;
    function GetComment: string; virtual; stdcall; abstract;
    function GetPage: string; virtual; stdcall; abstract;
    function GetGlyph: HICON; virtual; stdcall; abstract;
    function GetStyle: TExpertStyle; virtual; stdcall; abstract;
    function GetState: TExpertState; virtual; stdcall; abstract;
```

```
    function GetIDString: string; virtual; stdcall; abstract;
    function GetMenuText: string; virtual; stdcall; abstract;

    {  Запуск эксперта }
    procedure Execute; virtual; stdcall; abstract;
  end;
```

Это объявление класса выглядит несколько непривычно для программиста, привыкшего работать на языке C++, поскольку в нем использован синтаксис языка Object Pascal. Но, по существу, в нем имеются те же семантические элементы, что и в языке C++. Сравните эту версию объявления с версией на языке C++, приведенной в листинге 15.11. Обратите внимание: ни в той, ни в другой версии нет конструктора. Конструктор появится только в производном классе.

Листинг 15.11. Объявление класса `TIExpert` в среде C++Builder

```
class TIExpert : public TInterface
public:
    virtual String __stdcall GetName(void) = 0;
    virtual String __stdcall GetAuthor(void) =0 ;
    virtual String __stdcall GetComment(void) =0;
    virtual String __stdcall GetPage(void) = 0;
    virtual HICON __stdcall GetGlyph(void) = 0;
    virtual TExpertStyle __stdcall GetStyle(void) =0;
    virtual TExpertState __stdcall GetState(void) =0;
    virtual String __stdcall GetIDString(void) =0;
    virtual String __stdcall GetMenuText(void) = 0;
    virtual void __stdcall Execute(void) = 0;
} ;
```

В дальнейшем мы часто будем использовать в качестве базового класс `TIBCB5PluginBase`, объявление которого представлено в листинге 15.12. Это объявление находится в файле `IBCB5PluginBase.h` в папке `source\Chapter15\TBCB5PluginComponents` на прилагаемом к книге компакт-диске.

Листинг 15.12. Объявление класса `TIBCB5PluginBase`

```
class TIBCB5PluginBase
{
public:
    virtual char* __stdcall GetPluginName(void) = 0;
    virtual char* __stdcall GetPluginAuthor(void) = 0;
    virtual char* __stdcall GetMenuText(void) = 0;
    virtual HICON __stdcall GetPluginGlyph(void) = 0;
    virtual char* __stdcall GetPluginVersion(void) = 0;
    virtual bool __stdcall Execute() = 0;
    virtual char* __stdcall GetPluginCopyright(void) = 0;
    virtual void __stdcall DoneExpert(void) = 0;
} ;
```

Этот класс в дальнейшем мы будем использовать в качестве базового для всех классов подключаемых компонентов. Как и в классе TIExpert, все его методы объявлены с квалификатором virtual, и их нужно перегружать в производных классах. Производные классы подключаемых компонентов будут размещаться в модулях DLL, поэтому запустите C++Builder и вызовите мастер ***DLL Wizard***. (Для этого в меню выберите команду New, а затем на вкладке New — DLL Wizard.) Оставьте настройки, предлагаемые мастером по умолчанию, и щелкните на кнопке OK. Перед сохранением DLL откройте диалоговое окно Project Options и установите для модуля DLL расширение plg (будем считать его подходящим сокращением слова "plug-in" — подключаемый компонент). Сохраните проект. Теперь выберите в меню команду New и добавьте в проект новый файл заголовка. Этот файл нужно поместить в тот же каталог, что и прочие файлы проекта. В новый файл заголовка введите объявление класса TIBCB5PluginBase из листинга 15.12.

Опять откройте проект модуля DLL и добавьте в него объявление класса TDemoPlugin, представленное в листинге 15.13.

Листинг 15.13. Объявление класса TDemoPlugin

```
class TDemoPlugin : public TIBCB5PluginBase
{
private:
    HWND FParentApp;

public:

    TDemoPlugin(HWND ParentApp);
    ~TDemoPlugin();
    virtual char* __stdcall GetPluginName(void);
    virtual char* __stdcall GetPluginAuthor(void);
    virtual char* __stdcall GetMenuText(void);
    virtual HICON __stdcall GetPluginGlyph(void);
    virtual char* __stdcall GetPluginVersion(void);
    virtual TBCB5PluginType __stdcall GetPluginType(void);
    virtual char* __stdcall GetPluginCopyright(void);
    virtual bool __stdcall Execute();
    virtual void __stdcall DonePlugin(void);
} ;
```

В этом классе нигде не используется тип AnsiString, популярный среди разработчиков, работающих в среде C++Builder. Это сделано с одной целью — избежать необходимости обращаться к модулям DLL управления памятью, разработанным в Borland. Другой плюс использования строк, завершающихся нулем, в том, что это позволяет реализовать подключаемый компонент с помощью любого компилятора C++, например Microsoft Visual C++. Учтите, если в дальнейшем вам придется распространять классы VCL или функции, возвращающие значения типа AnsiString или принимающие аргументы типа AnsiString, то нужно включать в комплект поставки и модули DLL управления памятью, разработанные фирмой Borland.

Приведенное объявление класса практически не нуждается в пояснениях. В состав производного класса TDemoPlugin включен конструктор и деструктор, отсутствовавшие в объявле-

нии базового класса. Конструктор принимает один аргумент типа HWND. Через него передается дескриптор вызывающего приложения, который нужен в том случае, если в DLL используется класс, зависимый от родительского окна, например класс дочернего MDI-окна, окна экранной формы C++Builder и т.д.

В класс также добавлен метод DoneExpert(), который будет вызываться из основного приложения. Этот метод извещает приложение о предстоящем удалении объекта. Если в вызывающем приложении используется механизм передачи сообщений Windows, то в него добавляется невидимое окно, которому и передается это сообщение. Методы GetMenuText() и GetGlyph() позволяют передавать в вызывающее приложение информацию о том, как DLL представляется в меню или на панели инструментов в вызывающем приложении. В дальнейшем мы будем часто пользоваться методами Execute() и DonePlugin(). Реализацию остальных методов вы найдете в папке source\Chapter15\TBCB5PluginComponents на прилагаемом к книге компакт-диске. В листинге 15.14 представлен программный код методов Execute() и DonePlugin().

Листинг 15.14. Методы Execute() и DonePlugin() класса TDemoPlugin

```
bool __stdcall TDemoPlugin::Execute()
{
    ShowMessage("Plugin Activated!");
    return true;
}
//---------------------------------------------------------
void __stdcall TDemoPlugin::DonePlugin(void)
{
    if (this)
        delete this;
}
```

Как видите, компонент, который мы рассматриваем в качестве примера, ничего существенного не выполняет — при вызове метода Execute() он только выводит на экран сообщение. Но в реальном, а не учебном подключаемом компоненте именно в этом методе реализуются основные функциональные возможности компонента. Метод DonePlugin() вызывается из основного приложения и удаляет объект класса компонента из памяти с помощью оператора delete.

Нам еще понадобится инструмент для установки подключаемого компонента. Вызывающее приложение получит указатель на эту функцию с помощью функции Windows API GetProcAddress(), о которой мы уже не раз упоминали в этой главе. Функция RegisterPlugin() будет “дверью”, через которую подключаемый компонент сможет “выйти в мир”. Текст функции представлен в листинге 15.15. Она возвращает вызывающему приложению указатель на объект подключаемого приложения. Поскольку вызывающему приложению не известно, какой именно класс, производный от TIBCB5PluginBase, реализован в подключаемом компоненте, ему требуется указатель на тип void. В вызывающем приложении этот указатель будет приведен к типу базового класса TIBCB5PluginBase, который известен приложению, т.е. появляется возможность вызывать методы, реализованные в производном классе. Теперь модуль DLL с подключаемым компонентом фактически готов. Сохраните его файлы и выполните компиляцию.

Листинг 15.15. Функция `RegisterPlugin()`, которая создает объект класса подключаемого компонента

```
extern "C" __declspec(dllexport)
void* __stdcall RegisterPlugin(HWND ParentApp)
{
    // Создание объекта класса TDemoPlugin.
    thisPlugin = new TDemoPlugin(ParentApp);
    return thisPlugin;
}
```

Класс TBCB5PluginManager

Ниже представлена информация о классе `TBCB5PluginManager`, предназначенном специально для работы с рассматриваемыми в этой главе подключаемыми компонентами. Класс `TBCB5PluginManager` берет на себя всю "грязную" работу, связанную с загрузкой подключаемых компонентов и организацией доступа к его функциям через Windows API. Перечни свойств, методов и событий класса представлены, соответственно, в таблицах 15.1, 15.2 и 15.3.

Таблица 15.1. Свойства класса `TBCB5PluginManager`

Свойство	Описание
`PluginExtension`	Расширение имени загружаемого файла подключаемого компонента
`PluginFolder`	Папка, в которой находится подключаемый компонент
`Version`	Версия менеджера подключаемых компонентов
`PluginCount`	Количество загруженных подключаемых компонентов

Таблица 15.2. Методы класса `TBCB5PluginManager`

Метод	Описание
`LoadPlugin()`	Загружает подключаемый компонент
`LoadPlugins()`	Автоматически загружает все подключаемые компоненты, которые находятся в заданной папке
`UnLoadPlugin()`	Выгружает заданный подключаемый компонент
`GetLoadedPlugins()`	Возвращает список загруженных подключаемых компонентов

Таблица 15.3. События класса `TBCB5PluginManager`

Событие	Описание
`OnStartLoading`	Дает возможность пользователю разрешить или запретить загрузку подключаемого компонента
`OnFinishedLoading`	Извещает пользователя об имени только что загруженного подключаемого компонента

Программный код этого класса имеется на прилагаемом к книге компакт-диске в папке `source\Chapter15\TBCB5PluginComponents`. Вы можете самостоятельно проанализировать

тексты программ всех методов класса. Хочу обратить ваше внимание на несколько нюансов использования этого класса для управления работой подключаемых компонентов. Если вы хотите избежать повторной загрузки подключаемых компонентов, используйте значение `FileName` в обработчике события `OnStartLoad`. Для этого с помощью метода `GetLoadedPlugins()`получите список всех подключаемых компонентов, загруженных в текущий момент, а затем проанализируйте, нет ли в этом списке файла, заданного в `FileName`. Если таковой в списке встретится, установите значение `false` в `CanLoad` обработчика события `OnStartLoad`. Это заблокирует повторную загрузку компонента. В обработчике события `OnFinishedLoading` можно организовать добавление пунктов в меню или пиктограмм на панель инструментов приложения.

Некоторые нюансы программирования подключаемых компонентов

На этом мы завершаем рассмотрение программирования подключаемых компонентов. Конечно, многое из этой обширной темы осталось "за кадром". Хочу вам дать несколько советов по программированию подключаемых компонентов, основываясь на собственном опыте.

- Всегда используйте квалификатор `__stdcall` при объявлении экспортируемых из DLL методов.
- Обращайте внимание на соответствие типов в интерфейсе подключаемого компонента.
- Старайтесь, чтобы разрабатываемые вами подключаемые компоненты были как можно менее связаны со средой разработки. Только это обеспечит им широкое применение в сочетании с любыми приложениями.

Резюме

В этой главе мы рассмотрели методику программирования модулей DLL и создания подключаемых компонентов. Вы познакомились с тем, как организовать статическое и динамическое связывание DLL, экспортировать из DLL функции и классы, использовать объекты MDI-экранных форм, создавать пакеты.

Если вам потребуется более подробная информация, к вашим услугам всегда электронная справка C++Builder и справочник *Win32 Programmer's Reference for DLLs*.

Глава

16

Программирование COM-приложений

Айонел Муньоз

COM — это сокращение от *Component Object Model* (модель составных объектов). Технология программирования с использованием модели COM разработана фирмой Microsoft с прицелом на решение двух проблем, с которыми повсеместно сталкиваются программисты, работающие в операционной среде Windows. Эта модель предоставляет в распоряжение программиста спецификацию, на основе которой он может разрабатывать объекты, способные функционировать в различной языковой и операционной средах. Модель определяет способы взаимодействия клиентских приложений, работающих на одной машине, с приложением-сервером, функционирующим на другой машине. За последнее время издано множество литературы с описанием модели COM, а потому в этой книге предполагается, что читатель уже знаком с основными идеями COM-технологии. Если же они вам незнакомы, то на время отложите эту книгу и перечитайте, например, главу 13 в книге *C++Builder 4 Unleashed* (эту книгу Кента Райздорфа (Kent Reisdorph) опубликовало издательство *Sams Publishing*). Затем можно вновь вернуться к изучению материала в этой книге.

Те, кто знает, какая шумиха поднята вокруг операционных систем Kylix и Linux, могут задать себе (и нам) вопрос: "Зачем программистам осваивать COM-технологию, если существуют такие прекрасные операционные системы?" Я не отношу себя к ярым приверженцам Windows, но тем не менее считаю, что в мире персональных компьютеров эта операционная система еще довольно долго будет доминировать. А для того чтобы разработанные вами программные продукты были конкурентоспособны на рынке пользователей персональных компьютеров, вам непременно придется освоить сначала модель COM, затем модели DCOM и COM+.

На модели COM базируется все современная технология разработки приложения как набора взаимодействующих двоичных модулей, создаваемых в разных языковых средах разными разработчиками. В среде разработки C++Builder, которая на сегодняшний день является одной из самым популярных среди программистов, имеются все необходимые инструментальные средства для создания самых разнообразных COM-приложений. В этой главе я задавался целью представить читателю обзорный материал по этим инструментальным средствам, который он вряд ли сможет где-либо найти в таком концентрированном виде.

COM-серверы и COM-клиенты

В одном из моих любимых фильмов герой — очень "крутой мужик" — сходит с экрана и попадает в реальный мир. Он пытается разбить кулаком ветровое стекло автомобиля и с удивлением обнаруживает, что это больно. Оказывается, в реальном мире все обстоит далеко не так, как в мире воображаемом.

Среда C++Builder разработана для тех, кто живет в реальном мире, и в ней все сделано таким образом, чтобы столкновение с этим миром не причиняло боль программисту — C++Builder защитит его как стальные рукавицы защищают руки рыцаря.

После того как в состав C++Builder 5 была включена библиотека ATL 3.0, возможности создания с ее помощью COM-приложений — как клиентских, так и серверных — значительно возросли. Однако, как это часто случается с новыми программными продуктами, некоторые возможности этой библиотеки, мягко говоря, не очень хорошо документированы. Например, вопрос о том, как с помощью C++Builder реализовать в приложении обработку событий интерфейсов диспетчеризации, постоянно дебатируется в группах новостей Borland (адрес группы новостей —`nntp://forums.inprise.com/borland.public.cppbuilder.activex`). Правда, в электронной справке версии C++Builder 5 как раз этот вопрос освещен достаточно подробно, чего нельзя сказать о такой теме, как обработка событий пользовательских интерфейсов.

При создании и серверных и клиентских COM-приложений мастера C++Builder не предлагают помощи в разработке обработчиков событий и приходится все делать вручную, полагаясь только на собственное искусство.

На заметку

Если вы не знаете, что такое интерфейс диспетчеризации, то, скорее всего, вы проигнорировали рекомендации, сделанные в начале этой главы. Пожалуйста, познакомьтесь в какой-либо другой книге с основными концепциями модели COM, а затем вернитесь к изучению материала в этой книге.

Интерфейс диспетчеризации в действительности является не COM-интерфейсом, а командным интерфейсом для доступа к языкам написания сценариев. Интерфейсы диспетчеризации реализуются как производные от стандартного базового интерфейса `IDispatch`.

Изучив эту главу, вы научитесь возбуждать события обоих типов в серверных COM-приложениях, перехватывать их на стороне клиента и соответственно реагировать. Вы познакомитесь и с тем, как разрабатывать модули DLL для реализации пользовательских интерфейсов, — эта тема также не очень хорошо освещена в сопроводительной документации C++Builder.

Выходные интерфейсы и обработка событий

Если COM-объект должен "играть" различные роли в разных ситуациях, в нем, как правило, реализуется несколько интерфейсов. Клиенты вызывают методы этих интерфейсов, а сервер реагирует на эти вызовы в соответствии с тем, как интерфейсы реализованы.

Обычно COM-объект должен обладать способностью известить клиентов о том, что произошло нечто ожидаемое ими, например завершена довольно длительная процедура. Получив такое уведомление, клиенты могут извлечь результат. Механизм возбуждения извещающих событий избавляет клиентов от необходимости периодически опрашивать состояние сервера, дожидаясь завершения длительной процедуры.

COM-объект на стороне сервера взаимодействует с клиентскими приложениями через выходные интерфейсы, которые реализуются на стороне клиента. Клиентское приложение подключает свои интерфейсы к серверу, используя *точки подключения* (connection points).

На заметку

При желании можно получить достаточно исчерпывающую информацию о COM на сайте MSDN по адресу `http://msdn.microsoft.com/`.

События, порождаемые механизмом обработки точек подключения, тесно связаны между собой. Обновленная модель событий используется в новой версии COM, которая получила наименование COM+. Но и новая модель не исключает использование прежнего механизма обработки точек подключения, особенно в тех случаях, когда требуется обеспечить высокую скорость работы.

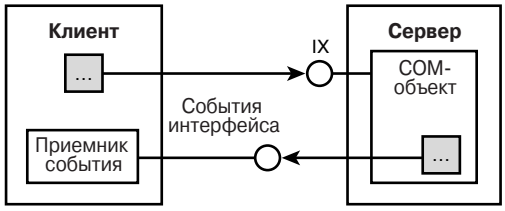


Рис. 16.1. Схема подключения COM-объекта на стороне сервера к приемнику события на стороне клиента

Для подключения на серверной стороне к объекту, который предоставляет выходной интерфейс, нужно на стороне клиента иметь объект, который реализует этот выходной интерфейс. Такой объект называется *приемником события* (*event sink*).

Если в составе COM-приложения на клиентской стороне имеется такой объект, его можно связать через точку подключения с COM-объектом на стороне сервера и запустить механизм обработки событий. На рис. 16.1 схематически представлена схема подключения COM-объекта на стороне сервера к приемнику события на стороне клиента.

А теперь займемся разработкой программного кода для серверного и клиентского COM-приложений.

Разработка COM-сервера

Поскольку хорошо сформулировать проблему — это наполовину ее решить, начнем с определения функций серверного приложения.

Сервер будет обслуживать клиентов, которые интересуются астрологическими гороскопами, а потому присвоим ему звучное наименование `Zodiac`. На сервер возлагаются две задачи.

- Он должен сформировать текст с именем знака зодиака, который соответствует дате рождения, введенной пользователем клиентского приложения. Эта информация может поступить от приложений, разработанных в любой языковой среде, например Visual Basic.
- Основываясь на дате рождения пользователя, сервер извлекает соответствующую информацию — прорицание, знак стихии (земля, огонь, вода, воздух), название планеты-покровителя. Эта информация помещается в заданную структуру, которую затем может обработать клиентское приложение, написанное на языке программирования, поддерживающем объектно-ориентированную модель (C++ или Delphi).

Исходя из такого перечня задач, нам придется реализовать на стороне сервера только один COM-объект. Объект будет иметь два интерфейса. Первый, более простой, является дуальным (dual), производным от базового интерфейса диспетчеризации `Idispatch`, и к нему можно получить доступ со стороны приложения, написанного на Visual Basic. Второй — более сложный пользовательский интерфейс, предназначенный для работы с клиентскими приложениями, желающими получить от сервера развернутую информацию.

На заметку

Дуальные интерфейсы используются для повышения производительности клиентских приложений, ориентированных на работу с серверами, при разработке которых применяются языки программирования со строгой грамматикой типов данных. Дуальный интерфейс позволяет использовать в самом клиентском приложении языки создания сценариев или язык Visual Basic.

Рекомендуется, если это возможно, не ориентироваться на использование дуальных интерфейсов, но несмотря на это разработчики мастеров C++Builder ориентировались по каким-то соображениям именно на них. Возможно, это связано с тем, что во многих организациях довольно широко используются приложения на Visual Basic и желательно обеспечить совместимость создаваемых COM-объектов с программами такого типа.

Создаваемый COM-объект будет возбуждать события для таких клиентов, и они смогут извлекать подготовленную информацию в асинхронном режиме. Пока сервер готовит информацию, клиентское приложение может выполнять другую задачу.

Выбор типа сервера

Серверное COM-приложение — это программа, которая содержит хотя бы один COM-компонент. Серверные приложения разделяются на *встроенные* (*inprocess*), *внешние* (*out-of-process*) и *удаленные* (*remote*).

Встроенный сервер всегда реализуется в виде модуля DLL. Компоненты встроенного сервера располагаются в том же адресном пространстве, что и клиентское приложение, которое их использует. Главное преимущество встроенных серверов — высокая скорость выполнения заявок клиента, а основной недостаток — возможность нарушения работоспособности всего клиентского приложения в случае аварии сервера, поскольку в памяти компьютера они не изолированы друг от друга.

Внешний COM-сервер реализуется в виде выполняемых EXE-файлов и занимает в процессе выполнения отдельное адресное пространство. Достоинства и недостатки внешнего сервера — это по сути недостатки и достоинства встроенного сервера (т.е. преимущества внешнего — это недостатки встроенного, а недостатки внешнего — преимущества встроенного).

Удаленный сервер размещается на отдельном компьютере, связанном по сети с компьютерами, на которых размещаются клиентские приложения. Они могут быть реализованы в виде модулей DLL (в этом случае используется специальный процесс, который образует оболочку DLL) или в виде выполняемых EXE-файлов.

Объекты, совместимые с моделью COM+/MTS, должны иметь форму DLL-модулей. В этом случае можно в полной мере реализовать преимущества этой технологии. Мы рассмотрим встроенный сервер.

Выбор модели потоков задач

Хотя в настройках мастеров COM-объектов C++Builder представлено несколько моделей потоков задач, фактически таких моделей только две. В новой терминологии эти модели называются *секционированной* (apartment) и *свободной* (free).

В самых общих чертах *секция объекта COM* (apartment) — это логическая часть потока, которая открывается вызовом функции `CoInitialize()` (или `CoInitializeEx()`), а закрывается вызовом `CoUninitialize()`. Основная цель, которая при этом преследуется, — ограничить область синхронизации. В каждом потоке, где предполагается использование COM-объекта, должна быть создана для него секция. Иногда это делает библиотека COM Library, иногда — нет. Это довольно сложный процесс, и его описание выходит за рамки данной книги.

Секционированные потоки имеют связанный с ними цикл обработки сообщений Windows. Все вызовы методов компонентов выполняются внутри секции потока, а потому являются синхронизированными. Элементы управления ActiveX обязательно используют секционированные потоки.

Поток в свободной модели может и не иметь собственного цикла обработки сообщений. Допускаются вызовы из любых потоков, а потому синхронизация возлагается на сам компонент, который получает сообщение.

В C++Builder используется прежняя терминология COM. В этой среде разработчику COM-сервера предлагается пять моделей организации потоков задач (вы встретитесь с ними в полях со списком Threading Model).

- **Single** (*однопотоковая*). Сервером не поддерживается множество потоков задач. Все заявки от клиентского приложения выстраиваются в очередь в главном потоке клиента. Эту модель использовать не рекомендуется.
- **Apartment** (**Single-Threaded Apartment**, *однопотоковая секционированная* или **STA**-модель). Каждый метод выполняется в потоке, который связан с определенным компонентом. Вызовы затем выстраиваются в очередь в потоке компонента, причем используется цикл обработки сообщений Windows в рамках отдельной секции потока. Каждый компонент может иметь собственную секцию (apartment) потока, что устраняет необходимость обращаться к главному потоку. Такая модель является более пригодной для масштабирования, чем однопотоковая (**Single**).

- **Free** (**Multi-Threaded Apartment** — многопотоковая секционированная или **MTA**-модель). Как было отмечено выше, эта модель позволяет вызывать любой метод в любом потоке в любой момент времени. Разработчик должен обеспечить механизм разрешения конфликтов при параллельном обращении к компоненту.
- **Both** (оба). Задается совместимость компонента с обоими описанными выше моделями. Системный механизм поддержки COM размещает компонент сервера в той же секции потока, что и компонент клиента, если только это возможно. Этот вариант настройки создаваемого компонента сервера наиболее предпочтителен, поскольку избавляет от множества забот, связанных с организацией маршалинга.
- **Neutral** (нейтральная). Это новый вариант модели потоков, который доступен только при использовании технологии COM+. Методы объекта могут одновременно вызываться из многих потоков, но синхронизация выполняется средствами поддержки COM+. Если же используется "чистый" механизм COM, то этот вариант по сути равнозначен варианту **STA**.

Модель организации потоков декларируется для COM-объекта статически, поддержка же декларированной модели обеспечивается программами реализации компонента. Заявление, что компонент поддерживает определенную модель организации потоков, означает, что компонент совместим с методикой синхронизации, специфицированной для этой модели.

Компонент COM-сервера, который мы будем рассматривать ниже, поддерживает модель **MTA**, хотя для большинства встречающихся на практике ситуаций вполне достаточно организовать поддержку модели **STA**. Мы выбрали вариант **MTA** специально для того, чтобы продемонстрировать в этой главе методику организации маршалинга. Модель **MTA** рекомендуется выбирать при создании специализированных серверов, от которых требуется высокая производительность, например серверов, работающих с аппаратными блоками.

Создание сервера

Для создания простого встроенного COM-сервера в среде C++Builder выполните следующие операции.

1. Запустите C++ Builder.
2. Выберите в системном меню File⇨New — на экране появится диалоговое окно New Items.
3. Выберите вкладку ActiveX и дважды щелкните на ярлыке ActiveX Library (рис. 16.2).

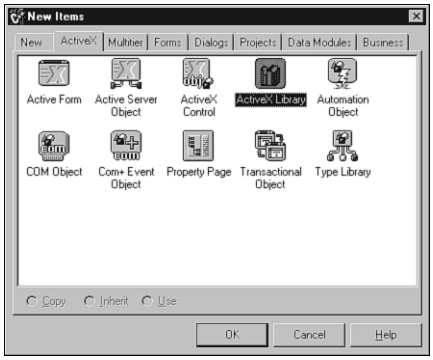


Рис. 16.2. Диалоговое окно New Items, в котором выбран ярлык ActiveX Library

4. Сохраните проект под именем `ZodiacServer`.

В этом проекте среда C++Builder создала заготовку встроенного COM-сервера.

На заметку Если вы собираетесь создать *внешний* COM-сервер, начните с создания обычного приложения.

Добавление COM-объекта

Теперь добавим в только что созданный проект программу реализации COM-объекта. Выполните следующие операции.

1. Выберите в системном меню File⇨New — на экране появится диалоговое окно New Items.
2. Выберите вкладку ActiveX. Поскольку нам требуется объект с дуальным интерфейсом (это позволит обращаться к нему из программ, написанных на языках формирования сценариев), дважды щелкните на ярлыке Automation Object. На экране появится диалоговое окно New Automation Object.

На заметку Если выбрать ярлык COM Object на этой вкладке, то по умолчанию будет создан COM-объект с пользовательским интерфейсом, вместо нужного нам дуального. Можете попробовать и такой способ создания COM-объекта сервера, но учтите, что он не сможет работать с клиентами, созданными на языках написания сценариев. Пояснения, которые будут приведены далее по ходу разработки, к такому серверу отношения не имеют.

3. В поле CoClass Name введите наименование объекта — `Zodiac`. При желании можете ввести в поле Description описание объекта.
4. Установите флажок Generate Event Support Code (генерировать программы поддержки событий) — этим дается знать C++Builder, что нужно автоматически сформировать программный код, необходимый для возбуждения событий выходного интерфейса диспетчеризации.
5. В поле со списком Threading Model выберите вариант `Free`. При этом, как отмечалось выше, настраивается поддержка многопотоковой секционированной модели потоков задач (**MTA**-модели). На рис. 16.3 показан вид диалогового окна New Automation Object после завершения настройки.

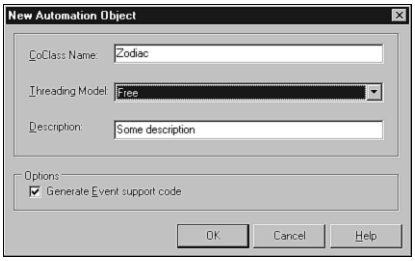


Рис. 16.3. Диалоговое окно New Automation Object, в котором выполнена настройка создаваемого компонента COM-сервера

После этого на экране откроется диалоговое окно редактора библиотеки типов ***Type Library Editor*** (***TLE***). Если этого не произойдет, выберите в меню команду View⇨Type Library. Окно редактора библиотеки типов представлено на рис. 16.4.

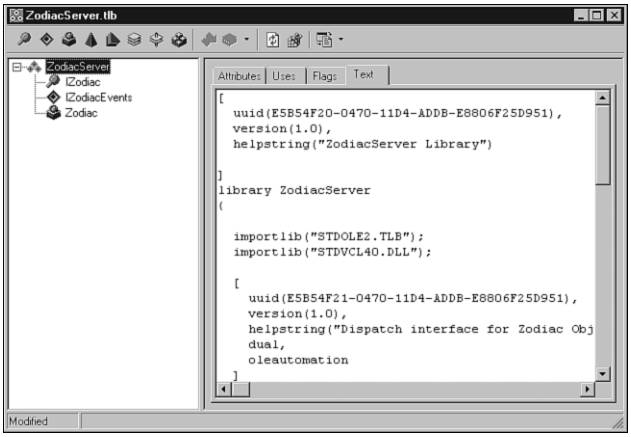


Рис. 16.4. Окно редактора библиотеки типов TLE, в котором показан созданный COM-объект

В левой панели этого окна показан *сопряженный класс* (coclass) `Zodiac` и два интерфейса — дуальный `IZodiac`, который является интерфейсом по умолчанию для создаваемого объекта сервера, и интерфейс диспетчеризации `IZodiacEvents`. Последний является выходным интерфейсом, который представляет события, возбуждаемые объектом.

На заметку

Классы, предназначенные для работы с COM-объектами, получили наименование *сопряженных классов* (*coclasses*). В сопряженном классе реализуется один или более интерфейсов. При просмотре программного кода библиотеки типов на языке *IDL* (*Interface Dafinition Langage*) определение сопряженного класса представляется в виде определения класса COM-объекта. Каждый сопряженный класс имеет атрибут `uuid`, который является уникальным идентификатором класса:

```
[
    uuid(4992BB45-FCA6-11D3-ADDB-BCAF427C7F50),
    version(1.0),
    helpstring("Zodiac Object")
  ]
  coclass Zodiac
  {
    [default] interface IZodiac;
    [default, source] dispinterface IZodiacEvents;
  } ;
```

В объявлении сопряженного класса перечислены все реализуемые им интерфейсы. Интерфейс, имеющий квалификатор `source` является *внешним* (*outgoing*), т.е. интерфейсом, который в действительности реализуется не в данном сопряженном классе, а клиентом, который связан с COM-объектом через точку подключения.

На вкладке Text этого диалогового окна выводится программный код библиотеки типов на языке IDL.

На заметку

Библиотека типов содержит описание внутренней структуры COM-сервера. Таким способом документируется любой COM-компонент. Обычно библиотека типов формируется в результате обработки исходного текста на языке IDL, хотя многие среды разработки, в частности и C++Builder, работают непосредственно с двоичным кодом библиотеки и не нуждаются в определении на языке IDL. Последний используется только для представления разработчику информации из библиотеки типов.

Библиотека типов распространяется вместе с COM-сервером в качестве ресурса или отдельно в виде файла с расширением `.TLB`. В некоторых организациях библиотеки принято не включать библиотеку типов в комплект поставки COM-сервера из соображений безопасности. Существуют определенные типы серверов, которые обязательно должны распространяться вместе с библиотекой типов, — это серверы автоматизации и элементы управления ActiveX.

Методика разработки COM-компонентов в среде C++Builder предполагает частое обращение к библиотеке типов. Среда C++Builder также организует связывание библиотеки типов с разработанным сервером. В процессе разработки COM-сервера пользователям C++Builder оказывает большую помощь редактор библиотеки типов ***Type Library Editor*** (***TLE***).

Добавим в интерфейс `IZodiac` один метод. Для этого выполните следующие операции.

1. Щелкните правой кнопкой мыши на узле `IZodiac` в левой панели диалогового окна и выберите в контекстном меню команду New⇨Method.
2. Предлагаемое по умолчанию наименование метода `Method1` замените именем `GetZodiacSign`.
3. В левой панели выделите элемент с именем метода и откройте справа вкладку Parameters.
4. Добавьте входные (`in`) параметры `Day` и `Month` типа `long`, которые будут представлять дату рождения клиента. Добавьте выходной (`out`, `retval`) параметр `Sign` типа `BSTR*`, через который будет возвращаться наименование знака зодиака, соответствующего введенной дате. На рис. 16.5 показано, что указатель мыши стоит на кнопке с многоточием в поле Modifier строки параметра `Sign`. После щелчка на этой кнопке можно установить модификаторы параметра.

На заметку

Параметры могут иметь один из модификаторов `in`, `out` или `in-out`. С помощью модификаторов при организации маршалинга определяется, передается ли значение параметра от сервера клиенту, от клиента серверу или можно передавать его в обе стороны.

Существует несколько правил обращения с параметрами в зависимости от вида модификатора. Для параметров с модификатором `out` сервер резервирует место в памяти, а клиент отвечает за ее высвобождение. Клиент полностью контролирует параметры с модификатором `in` — он организует как выделение памяти под этот параметр, так и ее высвобождение. Для параметров с модификатором `in-out` сервер может изменять размещение в памяти, но клиент по-прежнему отвечает за ее высвобождение.

Модификатор `retval` означает возвращаемое значение. Такой модификатор имеет смысл использовать для программ, формируемых на клиентской стороне оболочки. Мы еще увидим, как такой модификатор используется при создании клиентского приложения.

Существует много других видов атрибутов параметров, но их описание выходит за рамки этой книги.

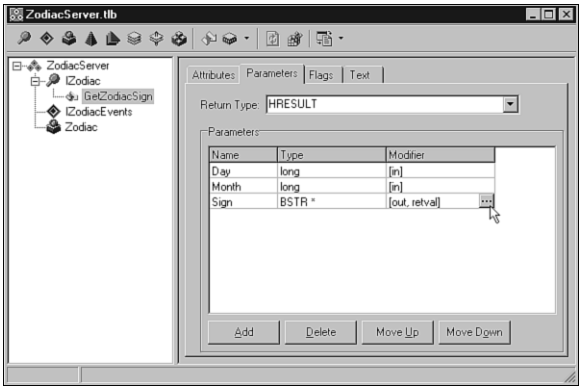


Рис. 16.5. Определение параметров метода с помощью вкладки Parameters диалогового окна редактора библиотеки типов

Метод `GetZodiacSign()`, как нетрудно догадаться, должен возвращать наименование знака Зодиака, соответствующего полученной от пользователя дате рождения.

Следуя той же методике, определите этот метод и его параметры. Метод будет асинхронно извлекать наименование знака Зодиака. Определение метода должно выглядеть следующим образом:

```
HRESULT _stdcall GetZodiacSignAsync([in] long Day,
        [in] long Month );
```

Обратите внимание на то, что метод `GetZodiacSignAsync()` не имеет выходного параметра `Sign`. Таким образом, для того чтобы извлечь сформированное наименование, нужно определить событие, которое и вернет его. Это событие должно быть возбуждено в момент, когда метод `GetZodiacSignAsync()` завершит выполнение операции.

Щелкните на узле интерфейса диспетчеризации `IZodiacEvents` в левой панели окна редактора и добавьте метод `OnZodiacSignReady()` (для этого нужно использовать ту же методику, с помощью которой ранее были добавлены методы в интерфейс `IZodiac`). Определение метода `OnZodiacSignReady()` должно иметь следующий вид:

```
HRESULT OnZodiacSignReady([in] BSTR Sign );
```

Учтите, что параметр `Sign` этого метода должен иметь модификатор `in`. Дело в том, что этот метод вызывается сервером, а программный код реализации метода размещается в приемнике события на стороне клиента. Поскольку с точки зрения клиента, этот параметр поступает от сервера, он является для него входным, т.е. должен иметь модификатор `in`. После выполнения всех этих операций окно редактора библиотеки типов должно выглядеть, как на рис. 16.6.

На этом этапе разработки у нас уже есть определения необходимых интерфейсов COM-объекта сервера. Мы только не показали, какой программный код при этом создан средой C++Builder. Теперь остается реализовать все означенные в интерфейсах функции, но это не вызовет затруднений.

Прежде чем продолжить работу, щелкните на кнопке Refresh Implementation, третьей справа на панели инструментов в окне редактора (после этого рекомендуется сохранить все файлы проекта). В составе проекта теперь имеется четыре файла: `ZodiacServer.cpp`, `ZodiacImpl.cpp`, `ZodiacServer_TLB.cpp` и `ZodiacServer_ATL.cpp`.

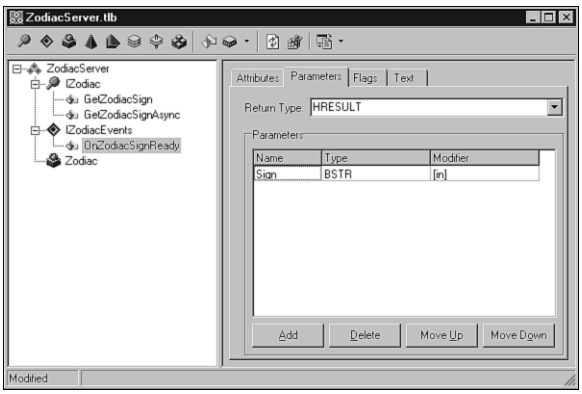


Рис. 16.6. Сопряженные классы COM-объекта в окне редактора библиотеки типов

Анализ сгенерированного программного кода

Прежде чем хвататься за разработку собственного программного кода, разберемся, что же в процессе всех описанных выше манипуляций "наваяла" среда C++Builder.

На заметку

В C++Builder для разработки COM-серверов используется библиотека ATL 3 (ATL — сокращение от `ActiveX Template Library`, библиотека шаблонов ActiveX). Фактически ATL представляет собой оболочку для разработки программного кода COM-объектов.

Файл `ZodiacServer_TLB.h` содержит текст определения всех ранее созданных программных компонентов на языке C++ (эти компоненты мы видели в окне редактора библиотеки типов). Фактически это перевод определений с языка IDL на C++. В самом начале файла `ZodiacImpl.h` вы увидите директиву включения другого файла, `ZodiacServer_TLB.h`, в котором имеются определения всех сопряженных классов:

```
#include "ZodiacServer_TLB.h"
```

На заметку

К файлу `ZodiacServer_TLB.h` мы еще вернемся при разработке клиентского приложения. Файлы `ZodiacServer_ATL.h` и `ZodiacServer_ATL.cpp` содержат, соответственно, программный код объявления и реализации объекта `_Module` (это объект класса `CComModule`). Его использует ATL для представления сервера и его глобальных данных.

В файле `ZodiacServer.cpp` находится точка входа DLL, глобальные функции регистрации сервера в средствах поддержки COM и карта объектов Object Map, с помощью которой сопряженные классы сервера отображаются на соответствующие генераторы классов.

Определение класса на языке C++ в файле `ZodiacImpl.h` выглядит так:

```
class ATL_NO_VTABLE TZodiacImpl :
  public CComObjectRootEx<CComMultiThreadModel>,
  public CComCoClass<TZodiacImpl, &CLSID_Zodiac>,
  public IConnectionPointContainerImpl<TZodiacImpl>,
  public TEvents_Zodiac<TZodiacImpl>,
```

```
  public IDispatchImpl<IZodiac, &IID_IZodiac,
              &LIBID_ZodiacServer>
```

Класс C++ `TZodiacImpl` — "двойник" сопряженного класса COM `Zodiac`. Он наследует от класса шаблона ATL `CComObjectRootEx`, в котором реализуются все операции подсчета ссылок и запросов базового интерфейса `IUnknown`, а также некоторые операции синхронизации, зависящие от выбранной модели потока задач. Поскольку при создании COM-сервера была выбрана модель **MTA**, созданный класс объявлен как производный от `CComObjectRootEx<CComMultiThreadModel>`.

Класс `CComCoClass` также входит в состав шаблонов ATL и поддерживает *генератор классов* (class factory) по умолчанию для объекта, определяет модель агрегирования и выполняет некоторые специфические функции обработки ошибок, о которых мы поговорим позже. Учтите, что наследование от класса `CComCoClass<TZodiacImpl, &CLSID_Zodiac>` означает, что идентификатор класса CLSID в вашем объекте уже сопоставлен с классом объекта. Эта ассоциативная связь декларируется в карте отображения объектов `ObjectMap` файла `ZodiacServer.cpp`:

```
BEGIN_OBJECT_MAP(ObjectMap)
  OBJECT_ENTRY(CLSID_Zodiac, TZodiacImpl)
END_OBJECT_MAP()
```

Таким способом устанавливается на уровне программного кода связь между объявленным сопряженным классом COM и его реализацией в сервере.

Класс `IDispatchImpl` — еще один шаблон, который используется при реализации дуальных интерфейсов, производных от базового интерфейса диспетчеризации `IDispatch`. Этот класс имеет дело с некоторыми нюансами реализации метода `Invoke()`, которые позволяют опережать вызовы автоматизации для определенного COM-интерфейса. В него же включена и реализация всех прочих сервисов интерфейса `IDispatch`. Класс `IDispatchImpl` объявлен в файле `atlcom.h` следующим образом:

```
template <class T, const IID* piid,
          const GUID* plibid = &CComModule::m_libid,
          WORD wMajor = 1, WORD wMinor = 0,
          class tihclass = CComTypeInfoHolder>
class ATL_NO_VTABLE IDispatchImpl : public T
{
     // Пропущено ...
} ;
```

Обратите внимание на то, что класс `IDispatchImpl` является производным от класса, заданного параметром шаблона `T`, который в нашем случае получает значение `IZodiac`. Таким образом, класс `IDispatchImpl<IZodiac, &IID_IZodiac, &LIBID_ZodiacServer>` при подстановке параметров шаблона не только реализует базовый набор функций интерфейса `IDispatch` применительно к нашему дуальному интерфейсу `IZodiac`, но также делает этот класс прямым наследником `IZodiac`. Использование механизма наследования — это основной метод, который позволяет в C++ указать интерфейс, реализуемый классом.

На заметку

Еще один способ реализации COM-интерфейсов предполагает использование агрегирования, но в данной книге мы его рассматривать не будем.

Класс IConnectionPointContainerImpl — еще один шаблон из состава ATL, который позволяет возложить на создаваемый COM-объект функции контейнера точек подключения и, таким образом, выполнить все процедуры, необходимые для возбуждения событий. Класс IConnectionPointContainerImpl обслуживает коллекцию объектов типа IConnectionPointImpl, каждый из которых представляет отдельный выходной интерфейс, поддерживаемый COM-объектом.

На заметку

При разработке COM-сервера вам не требуется знание деталей реализации шаблонов в ATL. Приведенных выше сведений вполне достаточно, но если вам захочется глубже познакомиться с принципами организации ATL 3, прочтите книгу *ATL Internals* (Chris Sells et al., ISBN: 0201695898, Addison-Wesley Publishing). Это лучшая из известных мне книг на эту тему.

Продолжим анализ подготовленного C++Builder программного кода. Класс TEvents_Zodiac в файле ZodiacServer_TLB.h сформирован по шаблону редактором библиотеки типов. На него имеет смысл обратить особое внимание, поскольку именно в этом классе возбуждаются события. Рассмотрим текст метода Fire_OnZodiacSignReady():

```
template <class T> HRESULT
TEvents_Zodiac<T>::Fire_OnZodiacSignReady(BSTR Sign)
{
  T * pT = (T*)this;
  pT->Lock();
  IUnknown ** pp = m_vec.begin();
  while (pp < m_vec.end())
  {
    if (*pp != NULL)
    {
      m_EventIntfObj.Attach(*pp);
      m_EventIntfObj.OnZodiacSignReady(Sign);
      m_EventIntfObj.Attach(0);
    }
    pp++;
  }
  pT->Unlock();
}
```

В этом методе последовательно просматривается массив клиентов, подключившихся к серверу. Каждый из клиентов информируется о том, что операция завершена. Это выполняется вызовом метода OnZodiacSignReady() объекта m_EventIntfObj.

Членами m_EventIntfObj являются объекты класса IZodiacEventsDisp, объявление которого в файле ZodiacZerver_TLB.h представлено в листинге 16.1.

Листинг 16.1. Файл ZodiacZerver_TLB.h, объявление класса IZodiacEventsDisp

```
template <class T>
class IZodiacEventsDispT : public TAutoDriver<IZodiacEvents>
{
public:
  IZodiacEventsDispT(){ }

  void Attach(LPUNKNOWN punk)
```

```
  {  m_Dispatch = static_cast<T*>(punk); }

  HRESULT  __fastcall OnZodiacSignReady(BSTR Sign/*[in]*/);

} ;
typedef IZodiacEventsDispT<IZodiacEvents> IZodiacEventsDisp;
```

Как видите, класс `IZodiacEventsDisp` является экземпляром класса шаблона `IZodiacEventsDispT`, который, в свою очередь, является производным от `TAutoDriver<IZodiacEvents>`. Класс `TAutoDriver` — это шаблон оболочки любого объекта автоматизации. `TAutoDriver` вызывает метод `Invoke()` интерфейса `IDispatch` внутри объекта автоматизации. Класс `TAutoDriver` взят не из библиотеки ATL, а из набора классов C++Builder, объявленных в файле `utilcls.h`.

Поскольку класс `TZodiacImpl` является производным от `TEvents_Zodiac`, то при необходимости возбудить событие можно обращаться к методу `Fire_OnZodiacSignReady()`. Классы библиотек ATL и EZ-COM (последнюю, по моему мнению, фирме Borland еще предстоит доработать) берут на себя всю "рутинную" работу по подготовке программного кода.

В листинге 16.2 приведено довольно запутанное объявление, взятое из файла `ZodiacImpl.h`, которое содержит несколько макросов и одну статическую функцию.

Листинг 16.2. Файл `ZodiacImpl.h`. Объявление атрибутов регистрации класса

```
// Данные, используемые при регистрации объекта
  //
  DECLARE_THREADING_MODEL(otFree);
  DECLARE_PROGID("ZodiacServer.Zodiac");
  DECLARE_DESCRIPTION("Some description");

  // Функция. которая вызывается при обновлении регистрации
  //
  static HRESULT WINAPI UpdateRegistry(BOOL bRegister)
  {
    TTypedComServerRegistrarT<TZodiacImpl>
    regObj(GetObjectCLSID(), GetProgID(), GetDescription());
    return regObj.UpdateRegistry(bRegister);
  }
```

Макросы, использованные в представленном фрагменте, служат для задания информации, фиксируемой в процессе регистрации в системном реестре Windows: тип модели потоков, идентификатор программы и описания сопряженных классов. Статическая функция `UpdateRegistry()` используется для регистрации, причем в процессе регистрации она оперирует перечисленной выше информацией и классом COM-объекта. Обращение к этой функции выполняется, когда возникает необходимость зарегистрировать COM-сервер в системном реестре Windows.

На заметку

В C++Builder регистрация происходит не так, как в Visual C++. В этом основное отличие технологии программирования ATL в этих средах.

Еще больше макросов используется при объявлении карты COM. Эта карта определяет связь между полученным вызовом QueryInterface() и запросами к классам IZodiac, Idispatch или IConnectionPointContainer:

```
BEGIN_COM_MAP(TZodiacImpl)
  COM_INTERFACE_ENTRY(IZodiac)
  COM_INTERFACE_ENTRY2(IDispatch, IZodiac)
  COM_INTERFACE_ENTRY_IMPL(IConnectionPointContainer)
END_COM_MAP()
```

Обратите внимание на то, что карта заполняется с помощью различных макросов. На практике чаще всего используется макрос COM_INTERFACE_ENTRY, который вставляет в карту один интерфейс. Он должен комбинироваться с производными от этого интерфейса. Как будет показано ниже, реализация методов интерфейсов также выполняется принудительно.

Так выглядит в файле карта точек подключения:

```
BEGIN_CONNECTION_POINT_MAP(TZodiacImpl)
  CONNECTION_POINT_ENTRY(DIID_IZodiacEvents)
END_CONNECTION_POINT_MAP()
```

Эта карта используется для создания коллекции идентификаторов каждого из выходных интерфейсов. Поскольку интерфейс диспетчеризации IZodiacEvents является выходным, в карту включается элемент с идентификатором этого интерфейса. В составе класса IConnectionPointContainerImpl имеется программный код, с помощью которого просматривается коллекция и определяется, реализована ли определенная точка подключения. Надеюсь, вы еще не забыли, что выходной интерфейс не имеет кода реализации в составе сервера — этот код содержится в программе клиентского приложения, связанного с сервером через точку подключения.

Разработка методов

Следующий этап — разработка методов, которые реализуют функциональные возможности сервера. Откройте в окне редактора программного кода файл ZodiacImpl.cpp . В нем вы увидите следующий текст:

```
////////////////////////////////////////////////////////////
// TZodiacImpl
STDMETHODIMP TZodiacImpl::GetZodiacSign(long Day,
               long Month, BSTR* Sign)
{
}

STDMETHODIMP TZodiacImpl::GetZodiacSignAsync(long Day,
                                        long Month)
{
}
```

Это — заготовка текста программ реализации методов, в которую нужно вставить содержательный программный код. Начнем с метода GetZodiacSign(). В этом методе нужно, на основе полученных значений аргументов Day и Month, сформировать строковое значение Sign (знак Зодиака). В том варианте программы, который мы здесь описываем, для выполнения этой операции используется запрос к таблице базы данных, но при желании вы можете реализовать ее и по-другому. Наша версия реализации метода выглядит следующим образом:

```
static TCOMCriticalSection CS;

STDMETHODIMP TZodiacImpl::GetZodiacSign(long Day,
                          long Month, BSTR* Sign)
{
  TCOMCriticalSection::Lock Lock(CS);

  ::GetZodiacSign(Day, Month, *Sign);

  return S_OK;
}
```

На заметку

Подробности операции "спрятаны" в функции `::GetZodiacSign()`, которая организует параметризированный запрос к модулю данных и извлекает из его таблицы соответствующий знак Зодиака. Полный программный код модуля данных вы можете найти на прилагаемом к книге компакт-диске, а сама функция будет описана позже в этой главе.

Поскольку COM-объект сервера настроен на модель потоков **MTA**, то для блокировки параллельных конкурирующих запросов на доступ к разделяемым ресурсам (в данном случае — базе данных) мы используем в этом методе объект класса `TCOMCriticalSection` (этот класс объявлен в файле `utilcls.h`). Один экземпляр класса `TCOMCriticalSection::Lock` выполняет функцию владельца критической секции в момент создания объекта, а при уничтожении объекта освобождает эту критическую секцию. Таким способом реализуется синхронизация, поскольку, как уже упоминалось выше, при использовании модели потоков **MTA** системные средства поддержки COM самоустраняются от синхронизации.

Фрагменты программного кода функции `::GetZodiacSign()` представлены в листинге 16.3.

Листинг 16.3. Файл `ZodiacImpl.cpp`, функция `GetZodiacSign()`

```
void GetZodiacSign(long Day, long Month, BSTR& Sign)
{
  // Сформировать параметризированный запрос.
  // Передать в качестве параметром значения Day и Month.
  // Активизировать запрос и т.д.:
     // Пропущено ...
  .
  .
  // Получить значения знака Зодиака из базы данных.
  ZodiacDataModule->QZodiac->First();
  WideString wstrSign =
    ZodiacDataModule->QZodiac->FieldByName(
        _T("Name"))->AsString.c_str();
  Sign = wstrSign.Detach();
}
```

Объект `WideString` является оболочкой вокруг `BSTR` и "владеет" строкой, но поскольку параметр `Sign` имеет модификатор `out`, нужно выделить под него память на стороне сервера и передать эту память клиенту. Это выполняется с помощью метода `detach()` объекта класса `WideString`.

Как видите, реализовать метод `GetZodiacSign()` не так уж сложно. Правда мы выпустили из поля зрения обработку ошибок.

Обработка ошибок

Зачастую программисты недооценивают важность организации в программе развитой системы обработки ошибок. В COM-приложениях, для которых характерен высокий уровень автономности компонентов (они ведь могут работать даже на разных машинах), эта функция является жизненно важной.

Кроме того, нужно учитывать, что в COM отсутствуют классы исключений, подобные тем, которые поддерживаются в модели CORBA. Поэтому первое, что должен сделать разработчик COM-приложений в среде C++Builder, — "упаковать" все методы интерфейсов в блоки try/catch. Иногда среда C++Builder самостоятельно генерирует эти блоки, но их следует включать в приложение и в тех случаях, когда C++Builder "считает" это излишним. Конечно, в самых простых методах, где вероятность возникновения исключительных ситуаций очень мала, можно было бы игнорировать это правило, но лучше этого не делать. Организация обработки исключительных ситуаций так важна потому, что исключение C++ не может выйти за пределы метода интерфейса. Представьте себе, что произойдет, если исключение C++ распространится на клиентское приложение Visual Basic.

Но если все исключительные ситуации обрабатываются только в методах реализации интерфейсов, то как передать информацию об ошибках на сторону клиента? Ведь у нас есть возможность возвращать только значение типа HRESULT. Для выполнения этих задач в COM-приложениях разработан специальный интерфейс ISupportErrorInfo:

```
interface ISupportErrorInfo : IUnknown
{
  HRESULT InterfaceSupportsErrorInfo([in] REFIID riid);
} ;
```

На сервере реализуются методы этого интерфейса, которые позволяют передать расширенную информацию об ошибках клиенту. Происходит это так: при перехвате исключительной ситуации вызывается COM-функция CreateErrorInfo(), которая создает объект ошибки. Эта функция возвращает интерфейс ICreateErrorInfo, который располагает методами заполнения объекта ошибки пользовательской информацией (например, в объект можно передать описание и идентификатор интерфейса, в котором обнаружен сбой).

После того как эта информация помещена в объект ошибки, ее можно извлечь оттуда, обратившись к методам интерфейса IErrorInfo этого объекта. Затем нужно вызвать COM-функцию SetErrorInfo(), которая связывает объект ошибки с текущим потоком.

На заметку

Для клиентского приложения задача существенно облегчается, если код ошибки возвращается методом интерфейса. Тогда оно может вызвать функцию GetErrorInfo() и получить развернутую информацию, представленную значением HRESULT. Функция GetErrorInfo() возвращает интерфейс IErrorInfo последнего созданного объекта ErrorInfo, ассоциированного с логическим потоком, из которого выполнен вызов. Более подробную информацию об этом можно получить в системе оперативной справки SDK, в разделе, посвященном функции GetErrorInfo().

К счастью, библиотека ATL берет на себя всю "черную" работу, и объекту сервера нужно только вызвать метод Error() класса CComCoClass. В листинге 16.4 представлен программный код реализации метода TZodiacImpl::GetZodiacSign(), в котором обрабатываются ошибки.

Листинг 16.4. Файл ZodiacImpl.cpp, метод TZodiacImpl::GetZodiacSign()

```
STDMETHODIMP TZodiacImpl::GetZodiacSign(long Day,
             long Month, BSTR* Sign)
{
  HRESULT hResult = S_OK;

  try
  {
    TCOMCriticalSection::Lock Lock(CS);

    ::GetZodiacSign(Day, Month, *Sign);
  }
  catch(Exception& e)
  {
    hResult = Error(e.Message.c_str(), IID_IZodiac, E_FAIL);
  }
  catch(...)
  {
    hResult = Error(_T("Catastrophic error: internal crash"),
              IID_IZodiac, E_UNEXPECTED);
  }

  return hResult;
}
```

Функция Error() создает и заполняет информацией объект ошибки, ассоциированный с интерфейсом IZodiac. Это выполняется при перехвате исключительной ситуации оператором catch. Код ошибки затем присваивается результату, который поступает в клиентское приложение.

Теперь все выглядит прекрасно, но вот одна неувязочка — мы забыли реализовать интерфейс ISupportErrorInfo! Но сейчас вы уже знаете, как это делается в среде C++Builder. Выполните следующие операции.

1. Скорректируйте объявление класса TZodiacImpl в файле ZodiacImpl.h, включив в перечень базовых классов ISupportErrorInfo:

   ```
   class ATL_NO_VTABLE TZodiacImpl :
     public CComObjectRootEx<CComMultiThreadModel>,
     public CComCoClass<TZodiacImpl, &CLSID_Zodiac>,
     public IConnectionPointContainerImpl<TZodiacImpl>,
     public TEvents_Zodiac<TZodiacImpl>,
     public IDispatchImpl<IZodiac, &IID_IZodiac, &LIBID_ZodiacServer>,
     public ISupportErrorInfo
   ```

2. Добавьте элемент для ISupportErrorInfo в карту COM:

   ```
   BEGIN_COM_MAP(TZodiacImpl)
     COM_INTERFACE_ENTRY(IZodiac)
     COM_INTERFACE_ENTRY2(IDispatch, IZodiac)
     COM_INTERFACE_ENTRY_IMPL(IConnectionPointContainer)
     COM_INTERFACE_ENTRY(ISupportErrorInfo)
   END_COM_MAP()
   ```

3. Объявите в составе класса методы интерфейса ISupportErrorInfo:

```
class TZodiacImpl :
     // Пропущено ...
{
     // Пропущено ...
  public:
   STDMETHOD(InterfaceSupportsErrorInfo)(REFIID riid);
} ;
```

4. Реализуйте метод InterfaceSupportErrorInfo() в файле ZodiacImpl.cpp:

```
STDMETHODIMP TZodiacImpl::InterfaceSupportsErrorInfo(
                                    REFIID riid)
{
  static const IID* arr[] =
  {
    &IID_IZodiac
  } ;
  for (int i=0; i < (sizeof(arr) / sizeof(arr[0])); i++)
  {
    if (InlineIsEqualGUID(*arr[i],riid))
      return S_OK;
  }
  return S_FALSE;
}
```

На заметку В реализацию метода InterfaceSupportsErrorInfo()в дальнейшем можно будет добавить и другие интерфейсы, если в этом возникнет необходимость.

Реализация метода, возбуждающего событие

В листинге 16.5 представлена реализация метода, асинхронно уведомляющего клиента о завершении операции. Рассмотренный в предыдущем разделе метод формировал такой же результат, но не возбуждал уведомляющее событие.

Листинг 16.5. Файл ZodiacImpl.cpp, метод TZodiacImpl::GetZodiacSignAsync()

```
STDMETHODIMP TZodiacImpl::GetZodiacSignAsync(long Day,
                                        long Month)
{
  try
  {
    TCOMCriticalSection::Lock Lock(CS);

    BSTR Sign = NULL;
    ::GetZodiacSign(Day, Month, Sign);
    Fire_OnZodiacSignReady(Sign);
    SysFreeString(Sign);
  }
```

```
  catch(Exception &e)
  {
    return Error(e.Message.c_str(), IID_IZodiac, E_FAIL);
  }
  catch(...)
  {
    return Error(_T("Catastrophic error: internal crash"),
                 IID_IZodiac, E_UNEXPECTED);
  }
  return S_OK;
} ;
```

В программном коде этого метода практически все операторы знакомы вам по предыдущим разделам. Вы уже знаете, что метод `Fire_OnZodiacSignReady()` возбуждает событие на стороне объекта сервера, которое вызывает соответствующий метод приемника события на стороне клиента.

Обратите внимание на то, как организовано выделение памяти. Сервер, как обычно, выделяет память под строку `Sign`, но в данном случае берет на себя и ее высвобождение, поскольку, с точки зрения клиента, `Sign` является входным параметром (имеет модификатор `in`). При программировании COM-объектов нужно очень внимательно следить за выделением/высвобождением памяти, чтобы предотвратить ее "утечку".

В своем нынешнем виде наш COM-объект способен получать и обрабатывать запросы и возбуждать события интерфейса диспетчеризации. Далее мы займемся дополнительным пользовательским интерфейсом, который позволяет извлекать расширенную информацию гороскопа.

Реализация пользовательского интерфейса

Приготовьтесь к тому, что практически вся работа по реализации пользовательского интерфейса выполняется "вручную" — вам придется иметь дело с определением интерфейса на языке IDL и программным кодом на языке C++. Несмотря на свой возраст, редактор библиотеки типов TLE в составе C++Builder все еще не "дорос" до столь сложных задач. Прежде чем приступать в объявлению пользовательского интерфейса, задумайтесь над тем, какие структурированные типы данных будут использоваться в нем для передачи аргументов.

Нам понадобится один структурированный тип, объявление которого на языке IDL выглядит следующим образом:

```
typedef struct tagTDetailedZodiacSign
{
  BSTR Sign;
  long House;
  BSTR Element;
  BSTR Planet;
  BSTR Details;
  BSTR Advice;
}  TDetailedZodiacSign;
```

Для того чтобы добавить в программу этот новый тип, выполните следующие операции.

1. Щелкните на кнопке New Record на панели инструментов редактора TLE (это пятая кнопка слева).
2. Присвойте записи имя `TDetailedZodiacSign`.

3. Добавьте в эту запись поля, которые перечислены выше в объявлении структуры. Для этого выделите имя структуры в левой панели окна редактора и щелкните на кнопке New Field на панели инструментов (эта кнопка указана курсором на рис. 16.7). Информация о каждом новом поле вводится в поля правой панели окна.

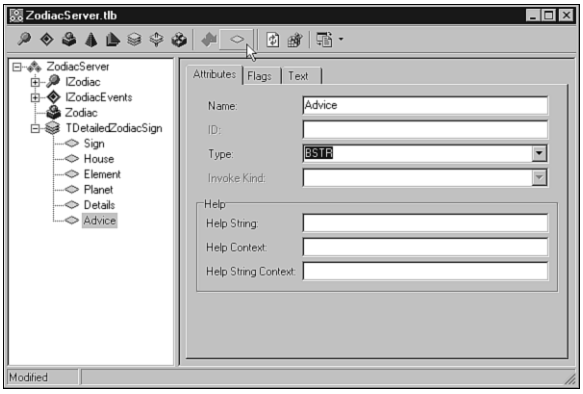


Рис. 16.7. Объявление нового типа в окне редактора библиотеки типов

Теперь добавьте в приложение новый интерфейс. Выполните для этого следующие операции.

1. Вызовите на экран окно редактора библиотеки типов TLE (команда меню View⇨Type Library).
2. Щелкните на кнопке New Interface на панели инструментов редактора TLE (это первая кнопка слева).
3. Присвойте интерфейсу имя `DetailedZodiac`.
4. Выделите узел интерфейса в левой панели окна редактора. В правой панели откройте вкладку Attributes и укажите в поле Parent Interface в качестве базового (родительского) для `IDetailedZodiac` интерфейс `IUnknown` вместо предлагаемого по умолчанию `IDispatch` (рис. 16.8).

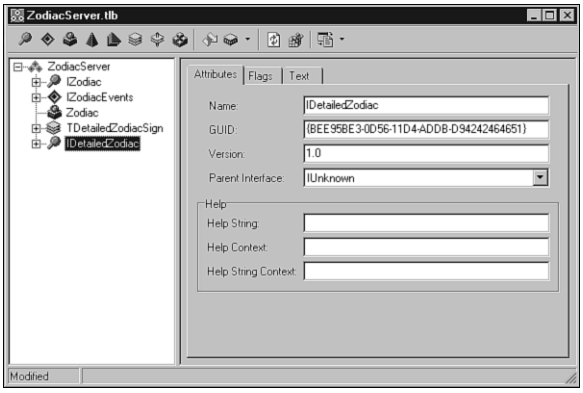


Рис. 16.8. Определение интерфейса `IDetailedZodiac` в окне редактора TLE

5. Откройте вкладку Indicators и сбросьте флажки Dual и Ole Automation. Если окажется, что эти флажки заблокированы, откройте вкладку Text и удалите атрибуты `oleautomation` и `dual`.

Теперь в этот интерфейс можно добавить два метода:

```
HRESULT _stdcall GetDetailedZodiacSign([in] long Day,
              [in] long Month,
              [out] TDetailedZodiacSign * DetailedSign );

HRESULT _stdcall GetDetailedZodiacSignAsync([in] long Day,
              [in] long Month );
```

Добавление методов в определение интерфейса выполняется по уже известной вам методике (вы имели возможность познакомиться с ней при определении интерфейса `IZodiac`). Не забывайте, что типом третьего параметра метода `GetDetailedZodiacSign()` является указатель на новый структурный тип. На рис. 16.10 показано, как выбирается тип параметра.

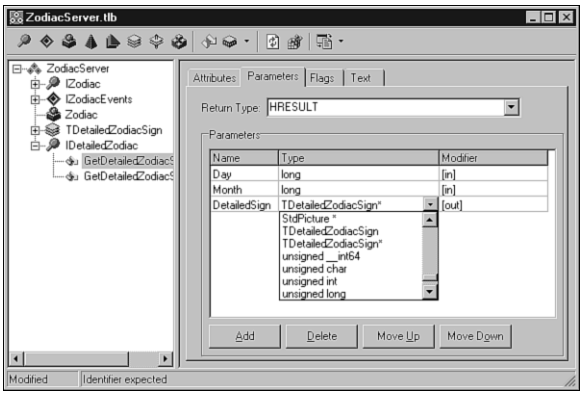


Рис. 16.10. Выбор типа параметра нового метода интерфейса

Теперь, когда сформировано определение интерфейса, можно с помощью редактора TLE указать сопряженный класс, который будет реализовать методы этого интерфейса.

1. В левой панели редактора выделите узел сопряженных классов `Zodiac`.
2. В правой панели откройте вкладку Implements. В этой вкладке выведен список всех интерфейсов, которые реализованы сопряженным классом `Zodiac`.
3. Щелкните на поле этой вкладки правой кнопкой мыши и выберите в контекстном меню команду Insert Interface. На экране появится диалоговое окно Insert Interface (рис. 16.9).
4. Выберите в этом диалоговом окне интерфейс `IDetailedZodiac` и щелкните на кнопке OK. После этого в список на вкладке Implements будет добавлен новый элемент `IDetailedZodiac`.
5. Сохраните результат проделанной работы — все файлы проекта.

Рис. 16.9. Диалоговое окно Insert Interface, в котором выделен интерфейс `IDetailedZodiac`

Откройте файл ZodiacImpl.h и посмотрите, что в нем изменилось. Во-первых, в глаза бросается, что в список базовых классов для TZodiacImpl добавлен новый макрос, который выглядит довольно необычно:

```
class ATL_NO_VTABLE TZodiacImpl :
  public CComObjectRootEx<CComMultiThreadModel>,
  public CComCoClass<TZodiacImpl, &CLSID_Zodiac>,
  public IConnectionPointContainerImpl<TZodiacImpl>,
  public TEvents_Zodiac<TZodiacImpl>,
  public IDispatchImpl<IZodiac, &IID_IZodiac, &LIBID_ZodiacServer>,
  DUALINTERFACE_IMPL(Zodiac, IDetailedZodiac)
{
   // Пропущено ...
} ;
```

Присутствие этого макроса показывает, что класс TZodiacImpl реализует дуальный интерфейс IDetailedZodiac. Но ведь IDetailedZodiac не является дуальным! Это ошибка редактора TLE, который включил в определение класса неправильную строку кода — в действительности в ней нужно указать IDetailedZodiac. Ниже показано, как должно выглядеть правильное определение списка базовых классов для TZodiacImpl:

```
class ATL_NO_VTABLE TZodiacImpl :
  public CComObjectRootEx<CComMultiThreadModel>,
  public CComCoClass<TZodiacImpl, &CLSID_Zodiac>,
  public IConnectionPointContainerImpl<TZodiacImpl>,
  public TEvents_Zodiac<TZodiacImpl>,
  public IDispatchImpl<IZodiac, &IID_IZodiac, &LIBID_ZodiacServer>,
  public IDetailedZodiac
{
   // Пропущено ...
} ;
```

Аналогичную ошибку допустил редактор TLE и при формировании карты COM:

```
BEGIN_COM_MAP(TZodiacImpl)
  COM_INTERFACE_ENTRY(IZodiac)
  COM_INTERFACE_ENTRY2(IDispatch, IZodiac)
  COM_INTERFACE_ENTRY_IMPL(IConnectionPointContainer)
  DUALINTERFACE_ENTRY(IDetailedZodiac)
END_COM_MAP()
```

Нужно заменить в этом списке макрос DUALINTERFACE_ENTRY макросом COM_INTERFACE_ENTRY. Вот как должны выглядеть карта COM для класса TZodiacImpl:

```
BEGIN_COM_MAP(TZodiacImpl)
  COM_INTERFACE_ENTRY(IZodiac)
  COM_INTERFACE_ENTRY2(IDispatch, IZodiac)
  COM_INTERFACE_ENTRY_IMPL(IConnectionPointContainer)
  COM_INTERFACE_ENTRY(IDetailedZodiac)
END_COM_MAP()
```

Таким образом, мы вынуждены были вручную скорректировать ошибки, которые допустил редактор библиотеки типов при переносе в файл введенной в окне информации. Теперь можно приступать к реализации методов пользовательского интерфейса IDetailedZodiac в файле ZodiacImpl.cpp.

В реализации метода GetDetailedZodiacSign() нет ничего нового, поэтому мы не будем рассматривать подробно соответствующий программный код. При желании вы всегда можете познакомиться с ним, скопировав с прилагаемого к книге компакт-диска.

Хотя метод GetDetailedZodiacSignAsync() очень похож на GetZodiacSignAsync(), в нем нужно организовать возбуждение события пользовательского интерфейса, которое мы еще не определили.

Возбуждение событий пользовательского интерфейса

Иногда у программистов возникает вопрос "А зачем в пользовательском интерфейсе возбуждать события?" Ответ прост: таким способом повышается скорость работы приложения. При вызове методов через интерфейс диспетчеризации появляется дополнительный уровень непрямой связи: нужно получить идентификатор интерфейса диспетчеризации этого метода, а затем вызвать метод Invoke() интерфейса IDispatch объекта. При вызове метода Invoke() передается довольно много параметров, в том числе идентификатор интерфейса диспетчеризации, тип вызова и пользовательские параметры, которым принудительно присваивается тип VARIANT. Такие среды разработки, как C++Builder, "скрывают" этот процесс от программиста (по крайней мере, частично), но дополнительные операции все равно остаются. Если разработчик заинтересован в максимальном быстродействии своего продукта, ему придется воспользоваться технологией возбуждения событий пользовательского интерфейса.

Объявление на языке IDL пользовательского выходного интерфейса, который нам предстоит реализовать, выглядит следующим образом:

```
[
  uuid(4992BB4F-FCA6-11D3-ADDB-BCAF427C7F50),
  version(1.0)
]
 interface IDetailedZodiacEvents: IUnknown
{
  HRESULT _stdcall OnDetailedZodiacSignReady(
                     [in] TDetailedZodiacSign * DetailedSign );
}
```

Как видите, объявление метода OnDetailedZodiacSignReady() очень похоже на объявление метода OnZodiacSignReady().

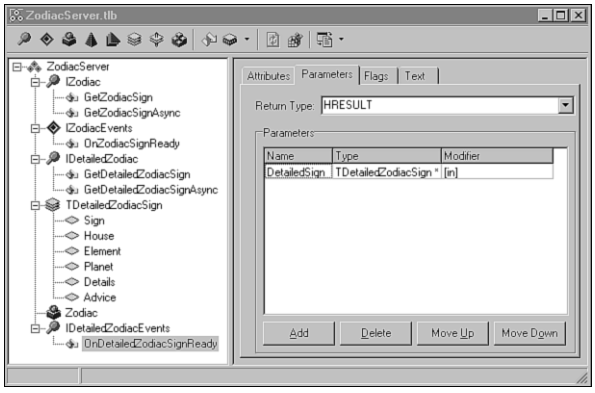


Рис. 16.11. Окно редактора библиотеки типов после включения в COM-объект интерфейса IDetailedZodiacEvents

Включение в программу интерфейса IDetailedZodiacEvents начнем с запуска редактора библиотеки типов (команды меню View⇨Type Library), а затем выполним ту же последовательность операций, что и при добавлении интерфейса IDetailedZodiac и его методов. После завершения этих операций окно редактора TLE должно выглядеть, как на рис. 16.11.

Теперь нужно указать статус нового интерфейса — будет ли он входным или выходным. Таковым он становится только по отношению к сопряженному классу. Для того чтобы включить в сопряженный класс Zodiac реализацию IDetailedZodiacEvents в качестве выходного интерфейса, нужно выполнить следующие операции.

1. Щелкните на узле сопряженного класса Zodiac в левой панели окна редактора TLE.
2. Откройте в правой панели вкладку Implements и щелкните на поле таблицы правой кнопкой мыши.
3. Выберите в контекстном меню команду Insert Interface — на экране откроется диалоговое окно Insert Interface.
4. Выделите в списке этого окна элемент IDetailedZodiacEvents и щелкните на кнопке OK
5. Щелкните на имени интерфейса IDetailedZodiacEvents правой кнопкой мыши и установите в контекстном меню флажок Source (рис. 16.12).
6. Сохраните результат проделанной работы — файлы проекта.

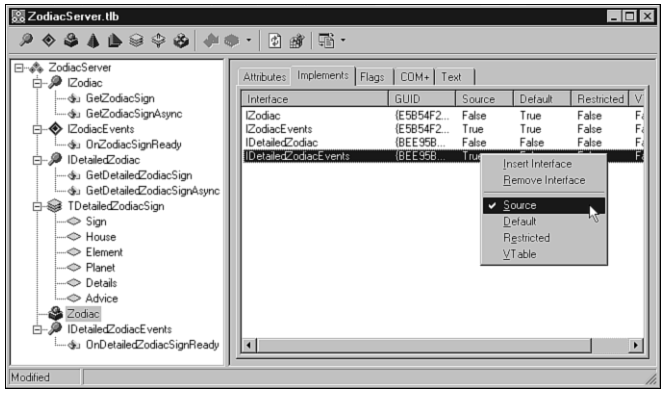


Рис. 16.12. Настройка IDetailedZodiacEvents в окне TLE как пользовательского выходного интерфейса сопряженного класса Zodiac

На этом процедура объявления интерфейса завершается. Объекты сопряженного класса Zodiac будут возбуждать события. которые объявлены в виде методов интерфейса IDetailedZodiacEvents.

Если теперь опять открыть в окне редактора программного кода файл ZodiacImpl.h, то в нем вы не увидите ничего нового. Среда C++Builder не формирует программный код поддержки событий пользовательского интерфейса. Поэтому его придется вводить вручную.

Если вы внимательно анализировали реализацию объявления TEvents_Zodiac в файле Zodiac_TLB.h, то сформировать аналогичный код для возбуждения события пользовательского интерфейса особого труда не составит. Один из возможных вариантов представлен в листинге 16.6.

Листинг 16.6. Файл ZodiacImpl.h, реализация класса TEvents_DetailedZodiac на основе шаблона

```
template <class T>
class TEvents_DetailedZodiac :
   public IConnectionPointImpl<T,
           &IID_IDetailedZodiacEvents,
           CComDynamicUnkArray>
{
   public:
   HRESULT Fire_OnDetailedZodiacSignReady(
                  TDetailedZodiacSign* DetailedSign)
   {
      T * pT = (T*)this;
      pT->Lock();
      IUnknown ** pp = m_vec.begin();
      while (pp < m_vec.end())
      {
         if (*pp != NULL)
         {
            CComQIPtr<IDetailedZodiacEvents,
                  &IID_IDetailedZodiacEvents> ptrEvents = *pp;
            if (ptrEvents != NULL)
               ptrEvents->OnDetailedZodiacSignReady(
                        DetailedSign);
         }
         pp++;
     }
     pT->Unlock();

     return S_OK;
   }
} ;
```

Основная особенность этого варианта реализации — в нем не используется объект оболочки m_EventIntfObj. Вызовы направляются прямо к интерфейсу IDetailedZodiacEvents, что обеспечивает более высокую скорость работы.

Назначение CComQIPtr<IDetailedZodiacEvents, &IID_IDetailedZodiacEvents> — организация запроса интерфейса IDetailedZodiacEvents со стороны подключенного к нему интерфейса IUnknown на стороне клиента. Запрос выполняется автоматически в конструкторе CComQIPtr, который является классом ATL. При удалении объекта класса CComQIPtr автоматически освобождается ссылка на интерфейс.

На заметку Шаблон класса TEvents_DetailedZodiac можно использовать в качестве отправной точки при организации поддержки любых событий пользовательского интерфейса.

Наступает последний этап работы. Нужно внести изменения в определение класса TZodiacImpl в файле ZodiacImpl.h. Выполните следующие операции.

1. В список базовых классов для TZodiacImpl добавьте TEvents_DetailedZodiac <TZodiacImpl>:

```
class ATL_NO_VTABLE TZodiacImpl :
  public CComObjectRootEx<CComMultiThreadModel>,
  public CComCoClass<TZodiacImpl, &CLSID_Zodiac>,
  public IConnectionPointContainerImpl<TZodiacImpl>,
  public TEvents_Zodiac<TZodiacImpl>,
  public IDispatchImpl<IZodiac, &IID_IZodiac,
                       &LIBID_ZodiacServer>,
  public IDetailedZodiac,
  public TEvents_DetailedZodiac <TZodiacImpl>
{
   // Пропущено ...
} ;
```

2. Внесите изменения в карту точек подключения — добавьте в нее новый элемент:

```
BEGIN_CONNECTION_POINT_MAP(TZodiacImpl)
  CONNECTION_POINT_ENTRY(DIID_IZodiacEvents)
  CONNECTION_POINT_ENTRY(IID IDetailedZodiacEvents)
END_CONNECTION_POINT_MAP()
```

В программный код метода GetDetailedZodiacSignAsync() добавьте вызов Fire_OnDetailedZodiacSignReady(). Этим вызовом клиент уведомляется о том, что асинхронно выполняемая процедура завершена, но на сей раз нужно использовать не интерфейс диспетчеризации, а пользовательский интерфейс. Новая версия метода GetDetailedZodiacSignAsync() представлена в листинге 16.7.

Листинг 16.7. Файл ZodiacImpl.cpp, метод TZodiacImpl:: GetDetailedZodiacSignAsync()

```
struct TDetailedZodiacSignImpl : public TDetailedZodiacSign
{
   TDetailedZodiacSignImpl()
   {
      Sign = NULL;
      Element = NULL;
      Element = NULL;
      Planet = NULL;
      Details = NULL;
      Details = NULL;
      Advice = NULL;
   }
   ~TDetailedZodiacSignImpl()
   {
      if (Sign != NULL)
        SysFreeString(Sign);
      if (Element != NULL)
        SysFreeString(Element);
      if (Planet != NULL)
        SysFreeString(Planet);
      if (Details != NULL)
```

```
        SysFreeString(Details);
      if (Advice != NULL)
        SysFreeString(Advice);
   }
} ;

STDMETHODIMP TZodiacImpl::GetDetailedZodiacSignAsync(
                long Day, long Month)
{
  try
  {
    TCOMCriticalSection::Lock Lock(CS);

    TDetailedZodiacSignImpl DetailedSign;
    ::GetDetailedZodiacSign(Day, Month, DetailedSign);
    Fire_OnDetailedZodiacSignReady(&DetailedSign);
  }
  catch(Exception &e)
  {
    return Error(e.Message.c_str(), IID_IDetailedZodiac, E_FAIL);
  }
  return S_OK;
} ;
```

Функция `::GetDetailedZodiacSign()`, имеющая глобальную область видимости, выделяет память для объекта `DetailedSign` структурного типа `TDetailedZodiacSign` и устанавливает значения его членов. Обратите внимание на класс оболочки `TDetailedZodiacSignImpl`, включенный специально для освобождения выделенной объекту памяти, поскольку в данном случае объект сервера должен распределять и утилизировать память, передаваемую клиенту вместе с параметром типа `in`. Об этом очень важно не забывать, а потому я опять останавливаю ваше внимание на механизме управления памятью.

На этом разработка COM-объекта сервера завершается. Можно скомпилировать его и зарегистрировать в системном реестре (выберите в системном меню C++Builder команду Run⇨Register ActiveX Server). Если все пройдет "без сучка и задоринки", на экране появится окно сообщения с текстом `Successfully registered ActiveX Server C:\<Your Path>\ZodiacServer.dll` (Зарегистрирован ActiveX-сервер C:\<*Путь к файлам проекта*>\ZodiacServer.dll).

Нам осталось разработать модуль DLL с "заместителем-заглушкой" (proxy/stub) для тех случаев, когда сервер будет использоваться в среде, не поддерживающей автоматизацию.

Разработка DLL с „заместителем-заглушкой"

Когда клиент и сервер размещаются на разных компьютерах или выполняются каждый в своем процессе, в работу включается механизм, который получил повсеместно наименование "маршалинг" (marshaling). В этот процесс вовлекаются два объекта: "заместитель" (или proxy-объект) и "заглушка" (или stub-объект).

Proxy-объект всегда размещается в том же адресном пространстве, что и клиентское приложение. В его обязанности входит "упаковка" параметров, передаваемых методу, и отправка

их серверу с помощью подходящего механизма передачи (обмен по сети, обмен между процессами в рамках операционной системы или обмен между потоками в рамках процесса).

Stub-объект размещается в адресном пространстве сервера. На этот объект возлагаются обратные функции: он должен распаковать параметры, поступившие от клиента, и предать их непосредственно методу сервера. Иногда этот процесс называют *unmarshaling* (т.е. маршалинг в обратном порядке). На рис. 16.13 схематически представлен процесс передачи параметров с помощью этого механизма.

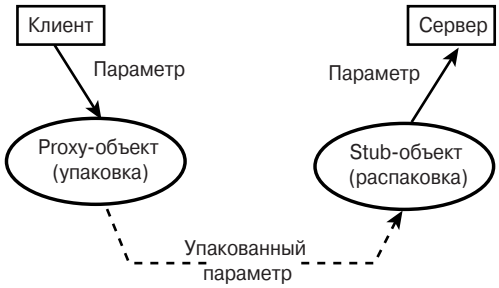


Рис. 16.13. Передача параметров между клиентом и сервером

На заметку

Известно три типа маршалинга. Маршалинг с использованием библиотеки типов (Type Library Marshaling) автоматически поддерживается COM для интерфейсов диспетчеризации и пользовательских интерфейсов OLE Automation через DLL-модуль `oleaut32.dll`. Стандартный маршалинг (Standard Marshaling) реализуется с помощью модулей DLL proxy/stub-объектов, которые нужно разработать самостоятельно. Стандартный маршалинг используется с пользовательскими интерфейсами, не входящими в категорию OLE Automation. Именно такой метод организации маршалинга мы и рассмотрим. Существует еще и третий тип — пользовательский маршалинг (Custom Marshaling), который применяется к объектам, в которых реализован интерфейс `Imarshall`.

Proxy- и stub-объекты связаны скорее не с клиентом и сервером, а с интерфейсами, поскольку с помощью этих объектов передаются параметры и возвращаемые значения интерфейсов.

Как вы уже знаете, модель COM базируется на интерфейсах. Когда регистрируется proxy/stub DLL, то в систему включаются средства реализации маршалинга для интерфейсов, которые объявлены в этом модуле. О существовании сервера такой DLL-модуль может даже не "знать". Каждый раз при первом обращении к COM-интерфейсу в контексте, требующем маршалинга, механизм поддержки COM просматривает системный реестр Registry в поиске proxy/stub-модуля DLL, который "взял бы на себя заботы" об этом интерфейсе. Это очень важно понимать, поскольку такая организация процесса позволяет включать объявления одних и тех же интерфейсов в разные COM-объекты и использовать вместе с ними те же proxy/stub-модули DLL.

Очень рекомендуется всегда разрабатывать proxy/stub-модули DLL для тех пользовательских интерфейсов, которые несовместимы с OLE Automation, даже если предполагается, что клиент вполне обойдется и без маршалинга.

Для создания proxy/stub-модуля DLL, работающего с приложением `ZodiacServer`, выполните следующие операции.

1. Откройте в среде C++Builder проект сервера.
2. Откройте окно редактора библиотеки типов — выберите в главном меню команду View⇨Type Library.
3. Щелкните на кнопке Export to IDL на панели инструментов окна редактора TLE (рис. 16.14).
4. Сохраните результат проделанной работы. Появится приглашение сохранить IDL-файл. Сохраните его под именем `ZodiacServer.idl`.
5. Закройте проект сервера.

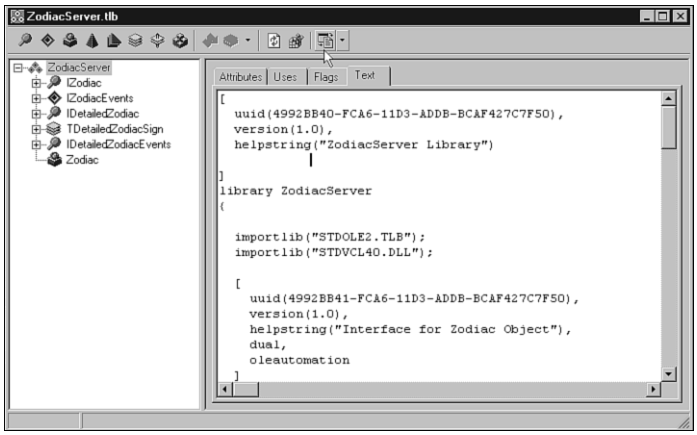


Рис. 16.14. Экспортирование из редактора TLE текста определения интерфейса на языке IDL

В листинге 16.8 представлено определение интерфейсов на языке IDL.

Листинг 16.8. Файл `ZodiacServer.idl`, сгенерированный редактором библиотеки типов

```
 [(Пропущено ...)]
library ZodiacServer
{
  importlib("STDOLE2.TLB");

  [(Пропущено ...)]
  interface IZodiac: IDispatch
  {
    [id(0x00000001)]
    HRESULT _stdcall GetZodiacSign([in] long Day,
                                   [in] long Month,
                                   [out, retval] BSTR * Sign );
    [id(0x00000002)]
    HRESULT _stdcall GetZodiacSignAsync([in] long Day,
                                   [in] long Month );
  } ;

  [(Пропущено ...)]
  dispinterface IZodiacEvents
```

```
  {
    properties:
    methods:
    [id(0x00000001)] HRESULT OnZodiacSignReady([in] BSTR Sign);
  } ;

  [(Пропущено ...)]
  typedef struct tagTDetailedZodiacSign
  {
    BSTR Sign;
    long House;
    BSTR Element;
    BSTR Planet;
    BSTR Details;
    BSTR Advice;
  }  TDetailedZodiacSign;

  [(Пропущено ...)]
  coclass Zodiac
  {
    [default] interface IZodiac;
    [default, source] dispinterface IZodiacEvents;
    interface IDetailedZodiac;
    [source] interface IDetailedZodiacEvents;
  } ;

  [(Пропущено ...)]
  interface IDetailedZodiac: IUnknown
  {
    [id(0x00000001)] HRESULT _stdcall GetDetailedZodiacSign(
               [in] long Day, [in] long Month,
               [out] TDetailedZodiacSign * DetailedSign );
    [id(0x00000002)] HRESULT _stdcall GetDetailedZodiacSignAsync(
               [in] long Day, in] long Month );
  } ;

  [(Пропущено ...)]
  interface IDetailedZodiacEvents: IUnknown
  {
    [id(0x00000001)] HRESULT _stdcall OnDetailedZodiacSignReady(
                      [in] TDetailedZodiacSign * DetailedSign );
  } ;
} ;
```

Чтобы можно было использовать этот файл для формирования программного кода proxy/stub-объектов, нам придется внести в него некоторые изменения. Выполните следующие операции.

1. Откройте файл `ZodiacServer.idl` в любом текстовом редакторе.
2. Перенесите из раздела `library` все пользовательские интерфейсы, которые не отмечены как совместимые с механизмом автоматизации, а остальные — удалите.

На заметку

Необходимость этого обусловлена тем, что компилятор IDL, разработанный в Microsoft (MIDL), формирует proxy/stub-код только для тех интерфейсов. которые определены вне раздела `library` исходного IDL-файла. Учтите, что дуальные интерфейсы, которые по определению совместимы с OLE Automation, переносить не нужно, поскольку по отношению к ним маршалинг организуется автоматически средствами поддержки COM.

3. Поместите в самом начале IDL-файла директиву `import "objidl.idl"`; в этом файле содержатся объявления стандартных типов OLE (листинг 16.9).

Измененная версия файла `ZodiacServer.idl` представлена в листинге 16.9.

Листинг 16.9. Измененная версия файла `ZodiacServer.idl`

```
// Стандартные определения COM:
import "objidl.idl";

  [(Пропущено ...) ]
  typedef struct tagTDetailedZodiacSign
  {
    BSTR Sign;
    long House;
    BSTR Element;
    BSTR Planet;
    BSTR Details;
    BSTR Advice;
  }  TDetailedZodiacSign;

  [(Пропущено ...) ]
  interface IDetailedZodiac: IUnknown
  {
    [id(0x00000001)] HRESULT _stdcall
           GetDetailedZodiacSign([in] long Day,
           [in] long Month,
           [out] TDetailedZodiacSign * DetailedSign );
    [id(0x00000002)] HRESULT _stdcall GetDetailedZodiacSignAsync(
                [in] long Day, [in] long Month );
  } ;

  [(Пропущено ... )]
  interface IDetailedZodiacEvents: IUnknown
  {
    [id(0x00000002)] HRESULT _stdcall OnDetailedZodiacSignReady(
                               [in] TDetailedZodiacSign * DetailedSign );
  } ;
```

Теперь этот IDL-файл можно “поручить заботам” компилятора MIDL, который запускается из командной строки. Компилятор сформирует несколько файлов исходного кода на языке C++, необходимых для формирования proxy/stub-модуля DLL.

Компилятор MIDL хранится в папке `C++Builder\Bin`. Для работы с ним удобнее всего пользоваться пакетным файлом команд. Вариант пакетного файла, которым пользуюсь я на своем компьютере, представлен в листинге 16.10.

Листинг 16.10. Пакетный файл команд для работы с компилятором MIDL

```
Rem  В тексте этого файла приведены пути на моем компьютере.
Rem  Вам придется ввести пути соответственно структуре файловой
Rem  системы на вашем компьютере.

@echo off

rem  Путь к включаемым idl-файлам, которые предполагается
rem  использовать в процессе компиляции.
set INCPATH=c:\progra~1\borland\cbuild~1\include\;
↳c:\progra~1\borland\cbuild~1\include\idl

rem Путь к препроцессору Borland C++ Builder
set CPPPATH=c:\progra~1\borland\cbuild~1\bin

rem Путь к компилятору MIDL
set MIDLPATH=c:\progra~1\borland\cbuild~1\bin

rem Вызов MIDL
@echo on

%MIDLPATH%\midl -ms_ext -I%INCPATH% -cpp_cmd%CPPPATH%\CPP32
↳ -cpp_opt "-P- -oCON -DREGISTER_PROXY_DLL -I%INCPATH%" %1
```

На заметку

Ключи настройки препроцессора компилятора подробно описаны в оперативной справке C++Builder. Более подробную информацию о компиляторе MIDL можно получить в MSDN по адресу `http://msdn.microsoft.com/`.

Сохраните свой пакетный файл под именем `makeIDL.bat`. После этого можно вызвать компилятор командой:

```
makeIDL ZodiacServer.idl
```

По завершении работы MIDL сформирует следующий набор файлов:

- `ZodiacServer.h`,
- `ZodiacServer_i.c`,
- `ZodiacServer_p.c`,
- `dlldata.c`.

Не забудьте сформировать самостоятельно `.def`-файл, который будет использоваться при регистрации модуля DLL и удалении его из реестра:

```
LIBRARY     ZODIACSERVERPS

  DESCRIPTION 'Zodiac Proxy/Stub DLL'

  EXPORTS
    DllGetClassObject PRIVATE
    DllCanUnloadNow PRIVATE
    DllRegisterServer PRIVATE
```

```
DllUnregisterServer PRIVATE
GetProxyDllInfo PRIVATE
```

Сохраните его под именем `ZodiacServerPS.def`.

Особого смысла разбираться в том, что "натворил" в перечисленных файлах компилятор MIDL, нет. Единственное, что следует помнить, — все эти файлы нужно включить в состав проекта DLL (их должно быть пять, включая `ZodiacServerPS.def` и `ZodiacServer.h`).

Теперь создадим в среде C++Builder новый проект. Выполните следующие операции.

1. Запустите C++Builder.
2. Выберите в главном меню команду File⇨New — появится диалоговое окно New Items.
3. Откройте вкладку New и щелкните на ярлыке DLL Wizard. Откроется диалоговое окно мастера DLL Wizard.
4. Установите переключатель C в зоне Source Type и щелкните на кнопке OK (рис. 16.15).
5. Удалите из списка файлов проекта `Unit1.c`, который автоматически вставляется в список средой C++Builder.
6. Включите в состав проекта файлы, созданные компилятором MIDL, и `.def`-файл.
7. Выберите в главном меню команду Project⇨Options на экране появится диалоговое окно Project Options.
8. Откройте вкладку Directories/Conditionals и введите в поле Conditional/Defines строку определения. Если вы собираетесь использовать proxy-объект в операционной системе Windows NT, эта строка должна выглядеть следующим образом:

   ```
   WIN32;_WINDOWS;_MBCS;_USRDLL;REGISTER_PROXY_DLL;
   ↳_WIN32_WINNT=0x0400
   ```

   Если proxy-объект предполагается сделать совместимым с Windows 9x, строка должна выглядеть так:

   ```
   WIN32;_WINDOWS;_MBCS;_USRDLL;REGISTER_PROXY_DLL;_WIN32_DCOM
   ```
9. Щелкните на кнопке OK.
10. Сохраните проект под именем `ZodiacServerPS.bpr`.
11. Скомпилируйте проект. В результате будет сформирован двоичный DLL-модуль.
12. Зарегистрируйте его как COM-сервер: `regsvr32 c:\<`*`путь на вашем компьютере`*`>\ZodiacServerPS.dll`. Для запуска регистрации воспользуйтесь командой Start⇨Run меню Windows.

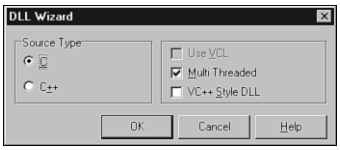


Рис. 16.15. Диалоговое окно мастера DLL Wizard

Voila! Теперь пользовательский интерфейс COM-объекта подкреплен модулем DLL, который и организует маршалинг. Созданный ранее сервер можно использовать с клиентским приложением, работающим в любой среде.

Для того чтобы продемонстрировать, как работает наш сервер, не хватает самой малости — клиентского приложения, которое нуждается в его услугах. Созданием такого приложения мы и займемся в следующем разделе.

Разработка клиентского COM-приложения

В этом разделе мы рассмотрим процесс создания клиентского COM-приложения, которое будет обращаться к созданному ранее COM-серверу. Процесс создания приложения начинается, как обычно, с создания в среде C++Builder нового проекта. В состав клиентского приложения должны входить графические средства диалога с пользователем, который с их помощью будет вводить дату своего рождения. Далее приложение должно обратиться к серверу и получить от него минимальную или расширенную информацию, соответствующую переданной дате рождения. Полученные данные нужно вывести на экран.

Экранная форма клиентского приложения изображена на рис. 16.16.

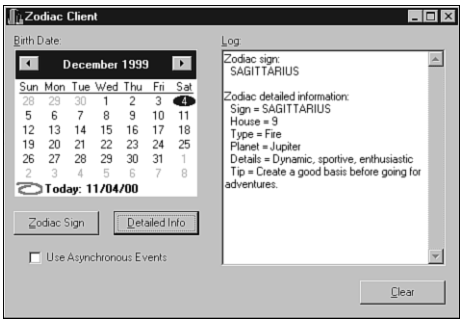


Рис. 16.16. Диалоговое окно клиентского COM-приложения

В правой части окна находится панель Log, в которой будет отображаться информация, полученная от сервера. Слева внизу имеется флажок Use Asynchronous Events (использовать асинхронные события). Если этот флажок установлен, то клиент будет вызывать асинхронные методы сервера, а если сброшен — синхронные.

Я думаю, что экранную форму вы можете сформировать самостоятельно. Если это вызывает затруднения, воспользуйтесь исходными файлами приложения `ZodiacClient`, которые имеются на прилагаемом к этой книге компакт-диске. В дальнейшем, я полагаю, тем или иным способом в проект `ZodiacClient` будет включено все необходимое для работы с экранной формой. Экранной форме присвойте имя `MainForm`, а ее модулю — имя `Main`. Далее мы сосредоточимся только на том, что имеет отношение к COM.

Импортирование библиотеки типов

Клиентское приложение должно располагать информацией о типах, которые экспортирует сервер (включая классы и интерфейсы), а она доступна только через библиотеку типов сервера. Поэтому следующий этап создания клиентского COM-приложения — импортирование библиотеки типов сервера.

Выполните следующие операции.

1. Вызовите в главном меню C++Builder команду Project⇨Import Type — на экране появится диалоговое окно Import Type Library (рис. 16.17).
2. Сбросьте флажок Generate Component Wrapper (формировать компонент оболочки). Эта настройка означает, что в дальнейшем вся работа будет вестись с кодом на языке C++, который сформирован на основании кода на языке IDL в библиотеке типов, а VCL-компоненты оболочки использоваться не будут.

3. В списке в верхней части окна Import Type Library выберите наименование библиотеки типов ранее созданного сервера, с которым будет работать это клиентское приложение. В данном случае нужно выбрать в списке строку `ZodiacServer Library (Version 1.0)`. Щелкните на кнопке Create Unit (формировать модуль). В проект клиентского приложения будет добавлен файл `ZodiacServer_TLB.cpp`.

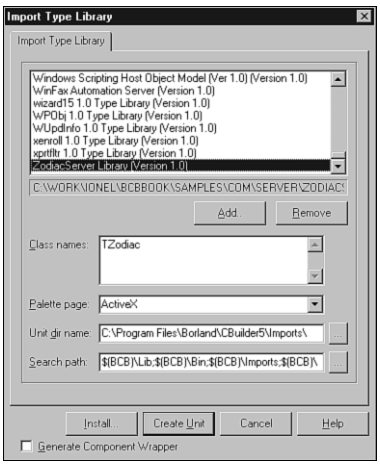


Рис. 16.17. Импортирование библиотеки типов сервера в клиентское COM-приложение

На заметку

В нашем случае можно было бы обойтись и без импортирования библиотеки типов сервера, поскольку мы создавали его самостоятельно, а потому располагаем всеми исходными файлами. Если вы сравните только что созданные файлы `ZodiacServer_TLB.cpp` и `ZodiacServer_TLB.h` с теми, которые имеются в составе проекта сервера, то увидите, что никаких отличий в них нет. Мы демонстрируем процедуру импортирования библиотеки типов сервера с одной целью: разработчик клиентского приложения не всегда располагает полным комплектом файлов сервера, а вот файл библиотеки сервера, зарегистрированного в системном реестре, всегда в вашем распоряжении. К тому же следует учесть, что сервер может быть разработан необязательно на языке C++, поскольку модель COM не привязана к определенному языку программирования. Связующим звеном между сервером и клиентами, разработанными на разных языках, и является библиотека типов.

Интерфейсы и сопряженные классы, определенные в библиотеке типов, теперь будут объявлены в файле `ZodiacServer_TLB.h`. В этом же файле вы обнаружите и несколько классов оболочек, которые являются частью C++Builder EZ-COM. (По мысли специалистов из Borland, EZ-COM должно означать "Easy COM" — COM, с которым легко работать, — но, к сожалению, утверждение, что с EZ-COM работать легко, мягко говоря, опрометчивое).

В листинге 16.11 представлены фрагменты файла `ZodiacServer_TLB.h`, в которых объявлены интерфейсы `IZodiac` и `IDetailedZodiac`.

Листинг 16.11. Файл `ZodiacServer_TLB.h`, объявление интерфейсов `IZodiac` и `IDetailedZodiac`

```
interface IZodiac : public IDispatch
{
public:
  virtual HRESULT STDMETHODCALLTYPE GetZodiacSign(
                  long Day/*[in]*/, long Month/*[in]*/,
                  BSTR* Sign/*[out,retval]*/) = 0; // [1]
  virtual HRESULT STDMETHODCALLTYPE
                  GetZodiacSignAsync(long Day/*[in]*/,
                  long Month/*[in]*/) = 0; // [2]
```

```
#if !defined(__TLB_NO_INTERFACE_WRAPPERS)
  BSTR __fastcall GetZodiacSign(long Day/*[in]*/,
                 long Month/*[in]*/)
  {
    BSTR Sign = 0;
    OLECHECK(this->GetZodiacSign(Day, Month, (BSTR*)&Sign));
    return Sign;
  }
#endif //   __TLB_NO_INTERFACE_WRAPPERS
} ;

   // Пропущено ...

interface IDetailedZodiac  : public IUnknown
{
public:
virtual HRESULT STDMETHODCALLTYPE GetDetailedZodiacSign(
                  long Day/*[in]*/, long Month/*[in]*/,
Zodiacserver_tlb::TDetailedZodiacSign*
               DetailedSign/*[out]*/) = 0; // [1]
  virtual HRESULT STDMETHODCALLTYPE GetDetailedZodiacSignAsync(
               long Day/*[in]*/, long Month/*[in]*/) = 0; // [2]
} ;
```

Как видно из представленного текста, интерфейс `IZodiac` является производным от `IDispatch`, а `IDetailedZodiac` — производным от `IUnknown`. Напомню, что один из этих интерфейсов является дуальным, а другой — пользовательским.

В составе интерфейса `IZodiac` присутствует и вторая версия метода `GetZodiacSign()`, являющаяся перегрузкой одноименного "оригинального" метода. Эта функция сгенерирована EZ-COM, поскольку параметр `Sign` декларирован как `retval`, т.е. как возвращаемое значение.

Вторая версия `GetZodiacSign()` возвращает строку, а не величину типа `HRESULT`. Внутри `GetZodiacSign()` вызывается "оригинальный" метод, результат выполнения которого обрабатывается макросом `OLECHECK`. Этот макрос возбуждает исключительную ситуацию, если вызов оригинального метода завершится неудачно.

На заметку

К сожалению, в макросе `OLECHECK` объединены функции доступа с функциями обработки ошибок, причем в последних не используются возможности `IErrorInfo`, которые могут быть реализованы на сервере (сервер `ZodiacServer` располагает такими возможностями). Поэтому сообщение, сформированное сгенерированной исключительной ситуации, остается невостребованным. Ему можно найти применение разве что в процессе отладки.

На компакт-диске, прилагаемом к этой книге, вы найдете файл `ComThrow.h`, а в нем — несколько макросов, которые можно использовать при разработке подсистемы обработки ошибок и исключений в COM-приложениях. Краткая инструкция содержится в комментариях в начале файла

Я бы рекомендовал заменять макрос `OLECHECK`, где бы вы его не встретили, одним из макросов из файла `ComThrow.h`, выбирая тот из них, который наиболее подходит к конкретной ситуации.

В клиентском приложении C++Builder обычно используется несколько шаблонов классов интерфейса вместо "чистых" интерфейсов. Такие специальные классы принято называть "интеллектуальными" (smart) интерфейсами. Одним из таких классов является TCOMIZodiac, представленный в листинге 16.12.

Листинг 16.12. Файл ZodiacServer_TLB.h, класс интерфейса TCOMIZodiac

```
template <class T /* IZodiac */ >
class TCOMIZodiacT : public TComInterface<IZodiac>,
                     public TComInterfaceBase<IUnknown>
{
public:
  TCOMIZodiacT() { }
  TCOMIZodiacT(IZodiac *intf, bool addRef = false) :
    TComInterface<IZodiac>(intf, addRef) { }
  TCOMIZodiacT(const TCOMIZodiacT& src) :
    TComInterface<IZodiac>(src) { }
  TCOMIZodiacT& operator=(const TCOMIZodiacT& src)
    {  Bind(src, true); return *this;}

  HRESULT     __fastcall GetZodiacSign(long Day/*[in]*/,
                                long Month/*[in]*/,
                                BSTR* Sign/*[out,retval]*/);
  BSTR        __fastcall GetZodiacSign(long Day/*[in]*/,
                                long Month/*[in]*/);
  HRESULT     __fastcall GetZodiacSignAsync(long Day/*[in]*/,
                                long Month/*[in]*/);
} ;
typedef TCOMIZodiacT<IZodiac> TCOMIZodiac;
```

Класс TCOMIZodiac представляет собой частную реализацию шаблона TCOMIZodiacT. Набор методов этого класса, в который входят оператор и инициализатор, существенно упрощает работу с интерфейсом.

Некоторые наиболее важные операторы в классах "интеллектуальных" интерфейсов наследуются от шаблона TComInterface. В этом шаблоне имеются операторы * и -> , с помощью которых можно получить доступ к "чистому" COM-интерфейсу. Кроме того, в составе TComInterface имеется собственный оператор присваивания, который освобождает текущий интерфейс, прежде чем присвоить объекту класса новый.

При инициализации объектов классов "интеллектуальных" интерфейсов используется экземпляр шаблона TCoClassCreatorT, который определен в файле utilcls.h (листинг 16.13).

Листинг 16.13. Файл Utilcls.h, шаблон класса TCoClassCreatorT

```
template <class TOBJ, class INTF, const CLSID* clsid,
         const IID* iid>
class TCoClassCreatorT : public CoClassCreator
{
public:
  static TOBJ    Create();
  static HRESULT Create(TOBJ& intfObj);
  static HRESULT Create(INTF** ppintf);
```

```
  static TOBJ    CreateRemote(LPCWSTR machineName);
  static HRESULT CreateRemote(LPCWSTR machineName,
           TOBJ& intfObj);
  static HRESULT CreateRemote(LPCWSTR machineName,
           INTF** ppIntf);
} ;
```

В оперативной справке C++Builder утверждается, что `TCoClassCreatorT` разработан для создания объектов, реализующих дуальные интерфейсы, но он прекрасно работает и с объектами, располагающими только пользовательскими интерфейсами.

Наиболее важным из методов класса `TCoClassCreatorT` является статический метод `Create()`, который вызывает функцию `::CoCreateInstance()`. Эта функция создает объект сопряженного COM-класса. Метод возвращает объект "интеллектуального" интерфейса, который является оболочкой "чистого" COM-интерфейса.

Класс `CoZodiac` образуется в результате конкретизации шаблона и объявлен в файле `ZodiacServer_TLB.h` следующим образом:

```
typedef TCoClassCreatorT<TCOMIZodiac, IZodiac,
                         &CLSID_Zodiac,  &IID_IZodiac> CoZodiac;
```

Создание и использование объекта COM-сервера

Для создания объекта сервера в клиентском приложении в состав класса экранной формы включается член типа `TCOMIZodiac`:

```
class TMainForm : public TForm
{
  TMonthCalendar *FCalendar;
  TButton *btnZodiac;
  TButton *btnDetailedZodiac;
  TCheckBox *chkAsync;
  TMemo *memLog;
     // Пропущено ...
private:
  TCOMIZodiac FZodiac;
} ;
```

Обратите внимание на то, что в состав класса включено объявление компонентов пользовательского интерфейса. Это позволит ссылаться на них в дальнейшем описании.

Создадим обработчик события `FormCreate()`:

```
FZodiac = CoZodiac::Create();
```

При запуске клиентского приложения на выполнение оно сразу же сформирует объект COM-сервера. Для этого даже не придется вызывать метод `CoInitialize()` — это сделает внутренний инициализатор. Кроме того, при завершении работы клиентского приложения будет автоматически вызван деструктор класса `TCOMIZodiac`, который освободит "захваченный" интерфейс сервера.

Теперь в экземпляре `FZodiac` можно вызывать любой метод `IZodiac`. В обработчик события `Click` кнопки Zodiac Sign включите программный код из листинга 16.14 (кнопка Zodiac Sign представлена в программе объектом `btnZodiac`).

Листинг 16.14. Файл `Main.cpp`, обработчик события `Click` кнопки Zodiac Sign

```
void __fastcall TMainForm::btnZodiacClick(TObject *Sender)
{
  TDateTime TheDate(FCalendar->Date);
  unsigned short year = 0;
  unsigned short month = 0;
  unsigned short day = 0;
  TheDate.DecodeDate(&year, &month, &day);
  if (!chkAsync->Checked)
  {
    // Используется класс оболочки!
    BSTR bstrSign = FZodiac.GetZodiacSign(day, month);
    WideString wstrSign = bstrSign;

    memLog->Lines->Add(_T("Zodiac sign:"));
    memLog->Lines->Add(_T("  ") + wstrSign);
    memLog->Lines->Add(_T(""));
  }
  else
  {
    OLECHECK(
      FZodiac.GetZodiacSignAsync(day,  month));
  }
}
```

На заметку В этом приложении, как и в большинстве других клиентских приложений C++, используется раннее связывание.

Если флажок Use Asynchronous Event в экранной форме клиентского приложения (это объект `chkAsync`) сброшен, этот обработчик считывает значение знака Зодиака, которое передается синхронным методом интерфейса. Если же флажок `chkAsync` установлен, вызывается асинхронный метод. Специальный обработчик должен перехватывать событие, возбужденное сервером по окончании выполнения запроса. Этот обработчик мы рассмотрим в следующем разделе.

Перехват событий, возбужденных интерфейсом диспетчеризации

COM-объект приложения должен содержать реализацию методов интерфейса диспетчеризации, объявленных в сопряженном классе сервера как выходные. Для разработки приемника события интерфейсов, производных от `IDispatch`, мы будем использовать шаблон класса `TEventDispatcher`, который находится в файле `utilcls.h`. Метод `InvokeEvent()` этого класса направляет вызовы сервера к соответствующим обработчикам событий. Объявление `InvokeEvent()` выглядит следующим образом:

```
// Перегрузить в производных классах для диспетчеризации событий.
virtual HRESULT InvokeEvent(DISPID id, TVariant* params = 0) = 0;
```

Класс TEventDispatcher также располагает методами, позволяющими подключать приемники событий к серверу (метод ConnectEvents()) и отсоединять их от сервера (метод DisconnectEvents()).

В приложении, которое рассматривается в этой главе, шаблон класса TEventDispatcher будет использован для создания класса, делегирующего обработку COM-события обработчику из VCL среды C++Builder (листинг 16.15). Объявление созданного класса включено в файл ZodiacSink.h.

Листинг 16.15. Файл ZodiacSink.h, реализация приемника события интерфейса диспетчеризации IZodiacEvents

```
// Объявление обработчика событий C++Builder VCL
typedef void __fastcall (__closure * TZodiacSignReadyEvent)
             (BSTR Sign);
//---------------------------------------------------------
// Сформировать класс, обслуживающий IZodiacEvents
class TZodiacSink :
   public TEventDispatcher<TZodiacSink, &DIID_IZodiacEvents>
{
protected:
  // Поле события
  TZodiacSignReadyEvent FOnZodiacSignReady;

  // Диспетчер событий
  HRESULT InvokeEvent(DISPID id, TVariant* params)
  {
      // Передача в обработчик VCL.
      if ((id == 1) && (FOnZodiacSignReady != NULL))
          // OnZodiacSignReady
          FOnZodiacSignReady(params[0]);
      return S_OK;
  }

  // Ссылка на источник события (используется указатель ATL)
  CComPtr<IUnknown> m_pSender;

public:
  __property TZodiacSignReadyEvent OnZodiacSignReady =
    {  read = FOnZodiacSignReady, write = FOnZodiacSignReady } ;

public:
  TZodiacSink() :
     m_pSender(NULL),
     FOnZodiacSignReady(NULL)  {   }

  // При удалении объекта автоматически
  // производится отключение от сервера.
  virtual ~TZodiacSink()   {  Disconnect(); }

  // Подключение к серверу.
```

```
  void Connect(IUnknown* pSender)
  {
     if (pSender != m_pSender)
        m_pSender = pSender;
     if (NULL != m_pSender)
       ConnectEvents(m_pSender);
  }

  // Отключение от сервера.
  void Disconnect()
  {
    if (NULL != m_pSender)
    {
      DisconnectEvents(m_pSender);
      m_pSender = NULL;
    }
  }
} ;
```

В методе `InvokeEvent()`, о котором речь шла чуть выше, проверяется значение ID интерфейса диспетчеризации, которое передается сервером (в данном случае оно равно 1). Зная это значение, можно перенаправить вызов соответствующему обработчику. Такова логика работы интерфейса диспетчеризации: определить ID, а затем делегировать обработку.

В классе `TMainForm` создадим член типа `TZodiacSink`:

```
TZodiacSink FZodiacSink;
```

Добавим в этот же класс и VCL-обработчик события типа `TZodiacSignReadyEvent`. Его реализация выглядит так:

```
void __fastcall TMainForm::OnZodiacSignReady(BSTR Sign)
{
    WideString wstrSign = Sign;
    memLog->Lines->Add(_T("Zodiac sign (ASYNCHRONOUS):"));
    memLog->Lines->Add(_T("  ") + wstrSign);
    memLog->Lines->Add(_T(""));
    wstrSign.Detach();
}
```

Как только поступает сообщение о событии, полученное значение фиксируется в памяти. Обратите внимание на то, что переменная `WideString wstrSign` отсоединяется от оболочки `BSTR` только в самом конце программы. Поскольку `Sign` является параметром с модификатором `in`, сервер высвобождает выделенную для этого параметра память. Класс оболочки `WideString` используется в данном случае только для удобства выполнения операции +, поскольку в этом классе оператор + перегружен.

Перейдем к обработчику события `FormCreate()`. Включим в него следующие операторы, обеспечивающие подключение к серверу:

```
  // Настройка свойства обработчика событий VCL
  FZodiacSink.OnZodiacSignReady = OnZodiacSignReady;
  // Подключение к серверу
  FZodiacSink.Connect(FZodiac);
```

Теперь все готово к работе. Скомпилируйте приложение и проверьте, как оно будет работать.

На заметку Прежде чем запускать приложение на выполнение, сформируйте с помощью BDE базу данных с псевдонимом ZODIAC. Как это делается, объяснено в сопроводительной документации BDE.

Запрос пользовательского интерфейса

Теперь рассмотрим, как использовать пользовательский интерфейс IDetailedZodiac. В файле ZodiacServer_TLB.h объявлено несколько типов указателей на классы "интеллектуальных" интерфейсов — по одному на каждую сервисную операцию сервера:

```
typedef TComInterface<IZodiac, &IID_IZodiac> IZodiacPtr;
typedef TComInterface<IZodiacEvents,
                &DIID_IZodiacEvents> IZodiacEventsPtr;
typedef TComInterface<IDetailedZodiac,
                &IID_IDetailedZodiac> IDetailedZodiacPtr;
typedef TComInterface<IDetailedZodiacEvents,
                &IID_IDetailedZodiacEvents>
                IDetailedZodiacEventsPtr;
```

Обратите внимание на то, что эти типы созданы на основе все того же шаблона класса TComInterface, который использовался нами ранее в качестве оболочки интерфейса.

Запрос к пользовательскому интерфейсу реализуется одном оператором:

```
// Неявный вызов QueryInterface:
IDetailedZodiacPtr DetailedZodiac = FZodiac;
```

Выполнение этого оператора приводит к неявному вызову функции QueryInterface() интерфейса IZodiac. После этого можно обращаться к методам IDetailedZodiac. Скорректируйте обработчик события OnClick кнопки Detailed Info (объект btnDetailedZodiac) — вставьте в него программный код из листинга 16.16.

Листинг 16.16. Файл Main.cpp, обработчик события Click кнопки Detailed Info

```
void __fastcall TMainForm::btnDetailedZodiacClick(
                TObject *Sender)
{
  TDateTime TheDate(FCalendar->Date);
  unsigned short year = 0;
  unsigned short month = 0;
  unsigned short day = 0;
  TheDate.DecodeDate(&year, &month, &day);
  // Неявный вызов QueryInterface:
  IDetailedZodiacPtr DetailedZodiac = FZodiac;
  if (!chkAsync->Checked)
  {
    TDetailedZodiacSignImpl DetailedSign;
    OLECHECK(
      DetailedZodiac->GetDetailedZodiacSign(day, month,
      &DetailedSign));
```

```
    memLog->Lines->Add(_T("Zodiac detailed information:"));
    memLog->Lines->Add(_T("  Sign = ") +
           AnsiString(DetailedSign.Sign));
       // Пропущено ...
    memLog->Lines->Add(_T("  Tip = ") +
           AnsiString(DetailedSign.Advice));
  }
  else
  {
    OLECHECK(
      DetailedZodiac->GetDetailedZodiacSignAsync(day, month));
  }
}
```

Этот метод очень похож на `btnZodiacClick()`, который был рассмотрен ранее. Операторы, представляющие наибольший интерес, выделены в листинге полужирным шрифтом. Думаю, вы разберетесь в этом методе и без нашей помощи.

Разработка приемника событий пользовательского интерфейса

На этот раз нам опять понадобится COM-объект клиентского приложения, в котором будет реализован пользовательский интерфейс `IDetailedZodiacEvents`. Эту работу придется выполнить вручную, но определенную помощь нам окажет ATL.

В файл `CustomEvents.h` нужно включить объявление класса оболочки приемников событий пользовательского интерфейса (листинг 16.17).

Листинг 16.17. Файл `CustomEvents.h`, шаблон класса `TCustomSink` для создания приемников событий пользовательского интерфейса

```
template <class Base, class Interface,
         const IID* piid = &__uuidof(Interface)>
class TCustomSink : public Base
{
private:
  CComPtr<IUnknown> m_ptrSender; // источник события
  DWORD m_dwCookie;

public:
  TCustomSink() :
    m_dwCookie(0) {  }

  virtual ~TCustomSink()  {  Disconnect(); }

  // Реализация IUnknown
  STDMETHOD_(ULONG, AddRef)() {  return 1; }
  STDMETHOD_(ULONG, Release)() {  return 1; }
  STDMETHOD(QueryInterface)(REFIID iid, void ** ppvObject)
    {  return _InternalQueryInterface(iid, ppvObject); }
```

```
public:
  // Методы подключения/отключения
  HRESULT __fastcall Connect(IUnknown* pSender)
  {
    HRESULT hr = S_FALSE;

    if (pSender != m_ptrSender)
    {
       m_ptrSender = pSender;
       if (m_ptrSender != NULL)
       {
         CComPtr<IUnknown> ptrUnk;
         QueryInterface(IID_IUnknown,
                reinterpret_cast<LPVOID*>(&ptrUnk));
         hr = AtlAdvise(m_ptrSender, ptrUnk, *piid, &m_dwCookie);
       }
    }

    return hr;
  }

  HRESULT __fastcall Disconnect()
  {
    HRESULT hr = S_FALSE;

    if ( (m_ptrSender != NULL) &&
           (0 != m_dwCookie) )
    {
      hr = AtlUnadvise(m_ptrSender, *piid, m_dwCookie);
      m_dwCookie = 0;
      m_ptrSender = NULL;
    }

    return hr;
  }
} ;
```

Как видите, этот шаблон класса реализует интерфейс `IUnknown`. Класс `TCustomSink` нужно применять в сочетании с `CComObjectRoot`, что позволяет воспользоваться преимуществами метода `QueryInterface()`. Механизм использования `QueryInterface()` поддерживается средствами ATL посредством обращения к `CComObjectRoot::InternalQueryInterface()` и карте интерфейсов. Аргумент `Base` в объявлении шаблона `TCustomSink` представляет пользовательский класс, производный от `CComObjectRoot`.

Хочу обратить ваше внимание и на другой момент. Мы используем ATL-функции `AtlAdvise()` и `AtlUnadvise()` в методах `Connect()` и `Disconnect()`. Эти функции осуществляют подключение к серверу и отключение от него.

Для того чтобы с помощью этого класса сформировать собственные приемники событий пользовательского интерфейса, выполните следующие операции.

1. Определите COM-класс на базе ATL, производный от `CComObjectRootEx` (или `CComObjectRoot`) и от `CComCoClass`.

2. Этот класс должен реализовать обработчики событий пользовательского интерфейса.
3. Объявите порождаемый приемник события — действительный класс приемника события — следующим образом:

   ```
   typedef TCustomSink<CMySink, &IID_IMyEvents> TMyCreatableSink;
   ```

   В данном случае класс `CMySink` должен реализовать пользовательский интерфейс `IMyEvents`.

В листинге 16.18 приведен полный текст файла `ZodiacCustomSink.h`, в котором реализован COM-касс и классы приемников событий интерфейса `IDetailedZodiacEvents`.

Листинг 16.18. Файл `ZodiacCustomSink.h`; класс `TDetailedZodiacSinkImpl`, в котором реализован приемник событий интерфейса `IDetailedZodiacEvents`

```
#if !defined(ZODIACCUSTOMSINK_H__)
#define ZODIACCUSTOMSINK_H__

#include <atlvcl.h>
#include <atlbase.h>
#include <atlcom.h>
#include <ComObj.HPP>
#include <utilcls.h>
#include "CustomSinks.h"
#include "ZodiacServer_TLB.h"

// Объявление обработчика событий C++Builder VCL
typedef void __fastcall (
          __closure * TDetailedZodiacSignReadyEvent)
    (TDetailedZodiacSign& DetailedSign);

//---------------------------------------------------------
// Создание класса, который получает IDetailedZodiacEvents
class ATL_NO_VTABLE TDetailedZodiacSinkImpl :
  public CComObjectRootEx<CComSingleThreadModel>,
  public CComCoClass<TDetailedZodiacSinkImpl, &CLSID_NULL>,
  public IDetailedZodiacEvents
{
public:
  TDetailedZodiacSinkImpl() :
    FOnDetailedZodiacSign(NULL)
  {
  }

  DECLARE_THREADING_MODEL(otApartment);

BEGIN_COM_MAP(TDetailedZodiacSinkImpl)
  COM_INTERFACE_ENTRY(IDetailedZodiacEvents)
END_COM_MAP()

protected:
  // Поле события
```

```
  TDetailedZodiacSignReadyEvent FOnDetailedZodiacSign;

public:
  __property TDetailedZodiacSignReadyEvent OnDetailedZodiacSign =
    {  read = FOnDetailedZodiacSign, write =
       FOnDetailedZodiacSign } ;

// IDetailedZodiacEvents
public:
  STDMETHOD(OnDetailedZodiacSignReady(
       TDetailedZodiacSign* DetailedSign))
  {
    if (FOnDetailedZodiacSign != NULL)
      FOnDetailedZodiacSign(*DetailedSign);
    return S_OK;
  }
} ;

typedef TCustomSink<TDetailedZodiacSinkImpl,
                    IDetailedZodiacEvents,
                    &IID_IDetailedZodiacEvents>
      TZodiacCustomSink;

#endif //ZODIACCUSTOMSINK_H__
```

Обратите внимание на то, что в этом файле используются те же классы ATL и макросы, которые применялись и при разработке сервера. Отличие в том, что в данном случае отпала необходимость в карте объектов, поскольку никакие объекты клиентом не создаются. По этой же причине не имеет значения идентификатор класса и мы используем `CLSID_NULL`. Класс `TDetailedZodiacSinkImpl` реализует только тот интерфейс, который используется сервером для возбуждения событий, и делегирует обработку событий VCL-обработчику типа `TDetailedZodiacSignReadyEvent` точно так же, как класс `TZodiacSink`, который рассматривался в предыдущем разделе.

Последним в файле является объявление класса `TZodiacCustomSink`, который и является классом приемника события:

```
typedef TCustomSink<TDetailedZodiacSinkImpl,
                    IDetailedZodiacEvents,
                    &IID_IDetailedZodiacEvents>
      TZodiacCustomSink;
```

В объявлении используется шаблон `TCustomSink`, который является оболочкой `TDetailedZodiacSinkImpl`.

Теперь можно приступить к разработке программного кода, использующего декларированный класс приемника событий. В классе экранной формы приложения определим член типа `TZodiacCustomSink`:

```
TZodiacCustomSink FZodiacCustomSink;
```

Добавьте в экранную форму VCL-обработчик события, фрагментарно представленный ниже:

```
void __fastcall TMainForm::OnDetailedZodiacSignReady(
                    TDetailedZodiacSign& DetailedSign)
```

```
{
    memLog->Lines->Add(
       _T("Zodiac detailed information (ASYNCHRONOUS):"));
    memLog->Lines->Add(_T("  Sign = ") +
        AnsiString(DetailedSign.Sign));

       // Пропущено ...

    memLog->Lines->Add(_T("  Tip = ") +
        AnsiString(DetailedSign.Advice));
    memLog->Lines->Add(_T(""));
}
```

Используя ту же методику, что и при организации подключения приемника событий интерфейса диспетчеризации, добавьте в метод `FormCreate()` следующие операторы:

```
// Настройка свойства обработчика событий VCL
  FZodiacCustomSink.OnDetailedZodiacSign = OnDetailedZodiacSignReady;
  // Подключение к серверу
  FZodiacCustomSink.Connect(FZodiac);
```

На этом работу можно считать завершенной. Мы создали клиентское приложение, которое позволяет воспользоваться всем спектром услуг сервера `ZodiacServer`. Скомпилируйте приложение, посмотрите, как оно работает, и можно идти за пивом!

Рекомендуемая литература

Общие вопросы программирования в среде C++ Builder

- C++Builder 4 Unleashed, Kent Reisdorph et al.; 1999, Sams Publishing; ISBN 0-672-31510-6

Технология COM/COM+

- Inside COM, Dale Rogerson; 1997, Microsoft Press; ISBN 1-57231-349-8
- Inside Distributed COM, Guy Eddon, Henry Eddon; 1998, Microsoft Press; ISBN 1-57231-849-X
- Understanding COM+, David S. Platt; 1999, Microsoft Press; ISBN 0-7356-0666-8
- Mastering COM and COM+, Ash Rofail, Yasser Shohoud; 1999, Sybex, Inc.; ISBN 0-7821-2384-8

Библиотека ATL

- ATL Internals, Brent Rector, Chris Sells; 1999, Addison Wesley; ISBN 0-201-69589-8

Программирование COM-приложений

- Эрик Хармон, Разработка COM-приложений в среде Delphi : Пер. с англ. — М.: Издательский дом “Вильямс”, 2000. — ISBN 5-8459-0074-3 (рус.)

Ресурсы Internet

- `nntp://forums.inprise.com/borland.public.cppbuilder.activex`
- `http://community.borland.com/cpp/`

- http://www.cetus-links.org/oo_ole.html
- http://www.techvanguards.com/
- http://msdn.microsoft.com/

Резюме

Материал, представленный в этой главе, поможет вам сделать первые шаги в программировании приложений, использующих технологию COM. Вы узнали о том, как в среде C++Builder создавать COM-сервер, реализовать дуальный и пользовательский интерфейсы, возбуждать события COM-объектов. С нашей помощью вы разработали специальный модуль DLL, обеспечивающий маршалинг между сервером и клиентом. В последних разделах главы была рассмотрена методика разработки клиентского COM-приложения. В следующей главе будет рассмотрена распределенная модель COM — DCOM, которая дает возможность использовать COM-объекты в сетевой среде.

Глава

17

Распределенные приложения, DCOM

Эдуардо Безерра

В предыдущей главе вы познакомились с моделью составных объектов COM и методами ее использования при создании приложений в среде C++Builder. Эти знания необходимы любому современному программисту, занятому разработкой программного обеспечения для операционной системы Windows. Все более широкий набор служб операционной системы предоставляется в виде COM-объектов, которые занимают то место, которое традиционно отводилось средствам интерфейса прикладных программ API.

Но обычные COM-объекты имеют один важный недостаток — их можно использовать только в пределах отдельного компьютера, т.е. и COM-сервер, и COM-клиент должны работать в рамках одной операционной системы. Нельзя разместить их на разных компьютерах и организовать общение через сетевую среду, т.е. создать в рамках технологии "обычной" COM распределенную программную систему.

Для того чтобы устранить этот недостаток, специалисты Microsoft разработали *распределенную модель* COM — DCOM (Distributed Component Object Model). Модель DCOM расширяет возможности COM, позволяя обращаться к COM-объектам в режиме удаленного доступа. В этой главе мы рассмотрим методику разработки распределенных приложений на основе модели DCOM, причем немало внимания будет уделено и новому для вас вопросу обеспечения безопасности, который для распределенных приложений является довольно существенным.

Базовые концепции DCOM

Распределенная модель COM — DCOM — разработана в Microsoft с тем, чтобы обеспечить взаимодействие между COM-объектами в режиме удаленного доступа. Это позволяет COM-серверу и COM-клиентам располагаться на разных компьютерах и связываться по сети.

По сути, механизм поддержки DCOM реализует те же задачи, что и механизм поддержки COM, плюс обеспечение безопасности приложений в сетевой среде. Говоря техническим языком, создание DCOM стало возможным в результате расширения возможностей механизма удаленного вызова процедур (Remote Procedure Call — RPC) и превращения его в Object RPC, что было впервые сделано в операционной системе Windows NT 4.0.

Поддержка DCOM реализована и в некоторых версиях UNIX, и в настоящее время эту модель можно рассматривать как средство реализации распределенных многоплатформенных приложений.

Несмотря на появление в последнее время других технологий, конкурирующих с DCOM в части поддержки распределенных вычислений, эта модель на сегодняшний день занимает ведущее место. Во многом это объясняется широким распространением операционной системы Windows NT на рынке средств для корпоративных приложений.

В модели DCOM используется концепция "прозрачности размещения" (Location Transparency), которая подразумевает, что можно, не изменяя программного кода выполнять приложение COM-сервера на разных компьютерах сети. При этом ни приложение-сервер, ни приложение-клиент не "знают", на каком компьютере фактически работает "собеседник". Они могут выполняться на одном компьютере, на разных компьютерах локальной сети, или даже на разных компьютерах, подключенных к глобальной сети, например к Internet. Приложение-клиент "узнает" о существовании сервера из данных в системном реестре, которые могут быть легко изменены.

Естественно, что за эту прозрачность размещения приходится платить. В данном случае это необходимость принимать дополнительные меры для обеспечения безопасности. В средствах поддержки DCOM эта задача решается в рамках сетевой системы доменов, уже давно развиваемой в Microsoft.

Клиентские и серверные DCOM-приложения должны следовать правилам аутентификации и авторизации. Это означает, что клиентское приложение должно "пройти" через определенную процедуру регистрации при попытке получить доступ к компьютеру, на котором выполняется серверное приложение. Сервер должен *аутентифицировать* (идентифицировать) клиента — проверить, действительно ли он является тем, за кого себя выдает, и *авторизировать* доступ — выяснить, имеет ли пользователь необходимые привилегии в системе.

Приложения DCOM могут полагаться на те параметры безопасности, которые хранятся в системном реестре Registry, или изменять эти параметры программно. Эти два подхода к реализации подсистемы безопасности получили наименования *декларативной безопасности* (*declarative security*) и *программируемой безопасности* (*programmatic security*).

Главной проблемой при программировании DCOM-приложений является как раз подсистема обеспечения безопасности, а потому именно ей мы и уделим наибольшее внимание в данной главе. Изучив ее, вы узнаете не только о том, какие требования предъявляет модель DCOM к средствам обеспечения безопасности, но и научитесь использовать имеющиеся в составе C++Builder 5 средства программирования для создания такой подсистемы.

Операционные системы семейства Windows и DCOM

В последних версиях операционных систем Windows — Windows 98, Windows NT 4.0 и Windows 2000 — поддержка модели DCOM стала стандартом, но ее не существует в наборе основных компонентов Windows 95. При желании вы можете загрузить программы DCOM95 с Web-сервера Microsoft по адресу `http://www.microsoft.com/com` или установить этот компонент в Windows 95 вместе с Internet Explorer 4/5.

На заметку

Если в локальной сети, к которой подключен ваш компьютер, имеется контроллер домена Domain Controller, то вам потребуется установить на своем компьютере такую конфигурацию Windows 95 или 98, которая обеспечит уровень безопасности пользователя (user-level). В противном случае вы не сможете выполнять аутентифицированные DCOM-запросы.

Операционные системы Windows 95 и 98 лучше всего использовать для выполнения клиентских DCOM-приложений, поскольку они не располагают возможностями динамически запускать и выполнять серверные приложения и не имеют собственной системы обеспечения безопасности.

Утилита DCOMCnfg

Утилита настройки конфигурации DCOM (DCOM Configuration Utility — файл `DCOMCnfg.exe`) представляет собой программу, с помощью которой можно изменять специфические настройки определенного DCOM-сервера или одновременно всех серверов такого типа, зарегистрированных в системном реестре.

После запуска программы `DCOMCnfg` на экране открывается ее диалоговое окно, в котором имеется четыре вкладки: Applications, Default Properties, Default Security и Default Protocols (рис. 17.1).

На заметку

Описание утилиты `DCOMCnfg` в этой главе опирается на ту версию, которая поставляется в составе Windows 2000 или Windows NT 4.0 с дополнением Service Pack 4 или более поздними дополнениями. Версия, поставляемая в составе Windows 9X, имеет небольшие отличия в интерфейсе.

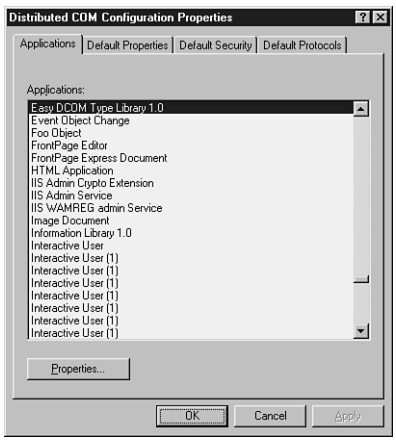


Рис. 17.1. Диалоговое окно программы `DCOMCnfg`*, вкладка* Applications

Настройка глобальных параметров подсистемы безопасности

На вкладке Applications (Приложения) выводится список всех серверов, которые зарегистрированы в ветви `AppID` системного реестра компьютера. Можно выбрать в этом списке любой элемент — сервер — и после щелчка на кнопке Properties (Свойства) получить доступ к его настройкам.

На вкладке Default Properties (Свойства по умолчанию) размещаются элементы управления, которые позволяют разрешить или заблокировать работу средств поддержки DCOM на данном компьютере или настроить системный уровень аутентификации и персонификации, применяемый по умолчанию (рис. 17.2).

На этой же вкладке можно разрешить или запретить работу COM Internet служб (COM Internet Services — CIS). CIS — это новый компонент модели DCOM, который позволяет клиентам и серверам взаимодействовать при наличии в системе прокси-серверов и брандмауэров. Более подробную информацию о CIS можно получить в системе справки MSDN по адресу `http://msdn.microsoft.com/library/backgrnd/html/cis.htm`.

Механизм поддержки аутентификации можно настроить на разные режимы работы: аутентификация выполняется при первом подключении клиента, при вызове метода или при передаче по сети каждого пакета данных между клиентом и сервером.

Выбрав в списке Default Authentication Level элемент `None`, можно отключить механизм аутентификации, действующий по умолчанию. Это рискованно, но позволит всем пользователям, независимо от личных привилегий, получать доступ к серверам на данном компьютере. Особую пользу из этого могут извлечь пользователи, подключающиеся к серверам через Internet.

Выбор в списке Default Impersonation Level определяет, может ли сервер по умолчанию проводить персонификацию подключающегося клиента. Если серверу предоставляется такая возможность, то существует подрежим, который определяет, будет ли сервер получать доступ к системным ресурсам “от имени” клиента. В этом списке можно выбрать следующие варианты настройки механизма персонификации клиента сервером:

- `Anonymous` — персонификация клиента не выполняется;
- `Identity` — сервер может персонифицировать клиента только для регистрации его имени;
- `Impersonate` — сервер может персонифицировать клиента и получить доступ к локальным системным ресурсам "от имени" клиента;
- `Delegate` — сервер может персонифицировать клиента и получить доступ как к локальным, так и к удаленным системным ресурсам "от имени" клиента. Этот режим возможен только в том случае, если в операционной системе Windows 2000 в качестве *провайдера системы безопасности* (Security Service Provider — SSP) используется Kerberos.

На вкладке Default Security (Безопасность по умолчанию) можно задать действующие по умолчанию системные настройки, определяющие режим доступа, запуска и настройки прав (рис. 17.3).

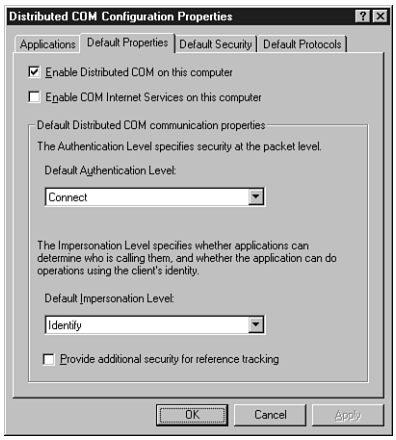


Рис. 17.2. Диалоговое окно программы `DCOMCnfg`, вкладка Default Properties

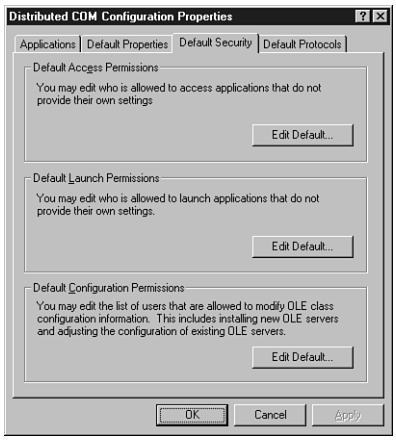


Рис. 17.3. Диалоговое окно программы `DCOMCnfg`, вкладка Default Security

После щелчка на кнопке Edit Defaults (Редактировать настройки по умолчанию) в зоне Default Access Permissions (Права доступа по умолчанию) можно перечислить тех пользователей, которым разрешен или запрещен доступ ко всем серверам, зарегистрированным на данном компьютере. Если вы считаете возможным предоставить по умолчанию доступ к серверам любым пользователям, укажите, что доступ открыт учетной записи `Everyone` (любой).

В зоне Default Launch Permissions (Права запуска по умолчанию) настраивается список пользователей, которые по умолчанию обладают правом загружать и выполнять программу любого сервера, зарегистрированного на данном компьютере. Если вы считаете возможным предоставить по умолчанию возможность запуска серверов любым пользователям, укажите, что запуск разрешен учетной записи `Everyone` (любой). Учтите, что эта настройка не может быть выполнена программно.

В зоне Default Configuration Permissions задается определенный тип доступа — `Read` (Чтение), `Full Control` (Полный контроль) или `Special Access` (Специальный вид доступа), — который фиксируется в ветви `HKEY_CLASSES_ROOT` системного реестра. С помо-

щью этой настройки можно эффективно управлять правами доступа пользователей в тем элементам в системном реестре, которые имеют отношение к функционированию механизма DCOM.

Вкладка Default Protocols (Протоколы по умолчанию) используется для настройки очередности и доступности сетевых протоколов, которые используются средствами поддержки DCOM (рис. 17.4). Выбрав подходящий сетевой протокол, можно настроить такой режим, который позволит DCOM работать через брандмауэр. Но до тех пор, пока вы не почувствуете себя хорошим специалистом в сетевых протоколах, лучше на этой вкладке никаких настроек не менять.

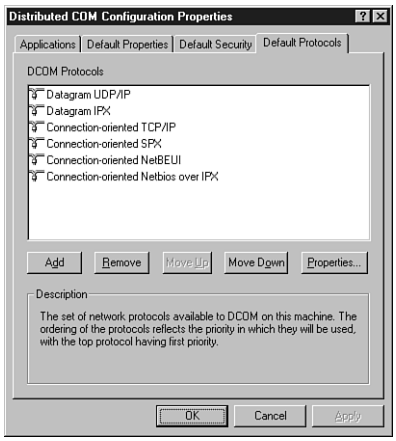


Рис. 17.4. Диалоговое окно программы `DCOMCnfg`*, вкладка* Default Protocols

Всегда помните, что настройки на вкладках Default Properties и Default Security распространяются на всю операционную системы. Непродуманное изменение этих настроек может привести к блокированию работы некоторых серверов, установленных на компьютере. Поэтому лучше всего изменять настройки, определяющие режим работы не всех серверов, установленных в системе, а тех, которые вам хорошо известны, причем каждого по отдельности.

Индивидуальная настройка параметров безопасности серверов

Утилита `DCOMCnfg` позволяет настраивать не только глобальные параметры безопасности, которые по умолчанию распространяются на все серверы, зарегистрированные в системном реестре, но и индивидуальные параметры каждого отдельного сервера. Это выполняется в диалоговом окне, которое открывается после выбора в списке на вкладке Applications имени сервера и щелчка на кнопке Properties. Примеры, которые вы увидите на иллюстрациях в этом разделе, относятся к серверу `Easy DCOM Type Library 1.0`.

На вкладке General выведено имя, под которым сервер зарегистрирован в системном реестре, его тип, уровень аутентификации и путь к файлу сервера на данном компьютере (рис. 17.5). Единственный параметр, который можно изменить на этой вкладке, — уровень аутентификации сервера. Его значение выбирается в поле со списком Authentication Level.

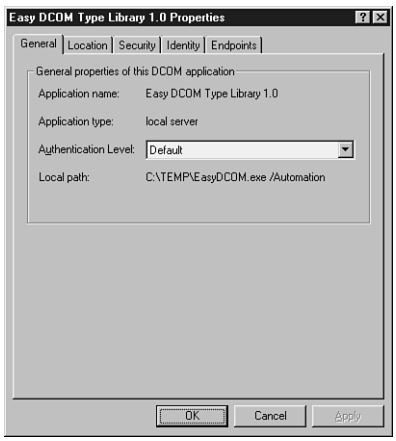


Рис. 17.5. Вкладка General *диалогового окна настройки параметров безопасности сервера*

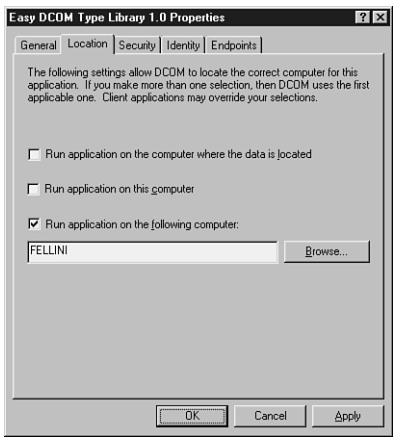


Рис. 17.6. Вкладка Location *диалогового окна настройки параметров безопасности сервера*

На вкладке Location можно указать, на каком компьютере будет выполняться серверное DCOM-приложение (рис. 17.6). Если, например, установить флажок Run Application on Following Computer и ввести в расположенное рядом поле сетевое имя компьютера, то серверное приложение будет выполняться именно на этом компьютере, а не на том, где выполняется настройка.

На вкладке Security можно выбрать наборы параметров безопасности, которые будут применяться к серверному приложению, — предусмотренные по умолчанию для всех серверов на данном компьютере или индивидуальные (рис. 17.7). Все индивидуальные параметры настраиваются точно так же, как и на вкладке Default Security (см. рис. 17.3).

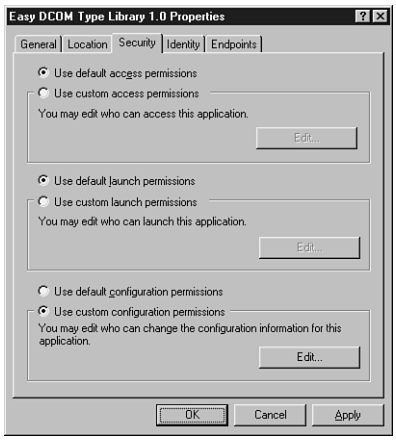


Рис. 17.7. Вкладка Security *диалогового окна настройки параметров безопасности сервера*

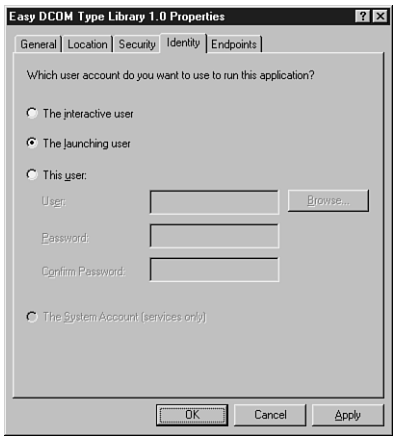


Рис. 17.8. Вкладка Identity *диалогового окна настройки параметров безопасности сервера*

На вкладке Identity задается учетная запись пользователя, от имени которой будет выполняться серверное приложение (рис. 17.8).

На этой вкладке имеется три переключателя.

- The Interactive User. Используется учетная запись текущего пользователя, зарегистрированного на том компьютере, где выполняется серверное приложение. Если в тот момент, когда поступает запрос на активизацию сервера, на этом компьютере никто не зарегистрирован, запрос не воспринимается. Этот режим используется в основном в процессе отладки, поскольку серверу может потребоваться доступ к рабочему столу зарегистрированного пользователя.
- The Launching User. Используется учетная запись пользователя, который запросил услуги объекта сервера. В этом режиме для каждого клиента создается своя копия процесса серверного приложения. Хотя этот режим и устанавливается по умолчанию, его рекомендуется избегать при работе с распределенными приложениями.
- This User. Используется учетная запись пользователя, специфицированного в размещенных рядом полях. При настройке параметров учетной записи можно использовать как уже существующую в операционной системе учетную запись, так и создать новую. В большинстве случаев именно этот режим является предпочтительным, поскольку заранее известно, какими привилегиями будет обладать серверное приложение в процессе выполнения.

Средства поддержки DCOM используют глобальный набор параметров безопасности, зафиксированных в системном реестре, для всех серверных DCOM-приложений, которые не имеют индивидуальной настройки. Если же для определенного сервера выполнена индивидуальная настройка, то соответствующие параметры имеют приоритет перед глобальными. Но в любом случае наивысший приоритет имеют настройки, установленные программно.

Следует всегда помнить о необходимости настройки прав запуска (Launch Permissions) для выбранных вами пользователей или групп пользователей. Если забыть об этом, удаленные пользователи могут не получить доступ к серверу.

Мы не будем касаться настроек, размещенных на вкладке Endpoints. Если вам понадобится какая-либо информация об этой вкладке, ее можно отыскать в сопроводительной документации программы `DCOMCnfg`.

Разработка DCOM-приложения

В этом разделе будет описан процесс разработки простенького DCOM-сервера и DCOM-клиента, которые мы будем использовать для экспериментов с моделью DCOM. У сервера довольно простая задача — в ответ на запрос клиента вернуть имя удаленного хост-компьютера и сформированные в нем текущую дату и время.

Серверное приложение будет оформлено в виде двоичного EXE-файла. Мы не будем оформлять приложение в виде модуля DLL, поскольку это потребует дополнительных усилий для организации внепроцессного режима выполнения. Приложение будет использовать маршалинг с помощью библиотеки типов, что позволит обойтись без proxy/stub-модуля DLL. О роли маршалинга в модели COM рассказано в главе 16.

Создание серверного приложения

Конечно, вы можете организовать систему подкаталогов для приложений как посчитаете нужным, но я на своем компьютере создал главный каталог для DCOM-приложений и в нем два подкаталога — `Server` и `Client`.

Полный набор файлов серверного DCOM-приложения. которое мы здесь рассматриваем, вы можете найти на прилагаемом к книге компакт-диске в папке `EasyDCOM\Server`.

Для создания серверного приложения выполните следующие операции.

1. Запустите C++Builder 5.
2. Выберите в главном меню команду File⇨New⇨Application.
3. Сохраните проект в папке `Server` под именем `EasyDCOM.bpr`, а файл `Unit1.cpp` переименуйте в `MainUnit.cpp`.
4. Откройте окно редактора главной экранной формы проекта.
5. Нажмите клавишу <F11> и вызовите на экран окно Object Inspector.
6. Свойству `Name` присвойте значение **`frmMain`**.
7. Скомпонуйте главную экранную форму так, чтобы она выглядела как на рис. 17.9.
8. Выберите в главном меню команду File⇨New.
9. В диалоговом окне New Items откройте вкладку ActiveX.
10. Дважды щелкните на ярлыке Automation Object. В ответ на экране откроется диалоговое окно New Automation Object.
11. В поле CoClass Name введите **`HostInfo`**.
12. В поле Description введите **`Easy DCOM Type Library 1.0`** и щелкните на кнопке OK.

Рис. 17.9. Экранная форма серверного приложения `EasyDCOM`

13. Откройте окно редактора библиотеки типов TLE (для этого следует выбрать в главном меню команду View⇨Type Library).
14. В левой панели выберите узел интерфейса `IHostInfo`.
15. Щелкните на кнопке New Property на панели инструментов и выберите из меню **`Read Only`**.
16. Замените предлагаемое по умолчанию имя свойства `Property1` на **`Info`**.
17. В правой панели откройте вкладку Parameters и установите в колонке Type значение `BSTR*` (рис. 17.10).

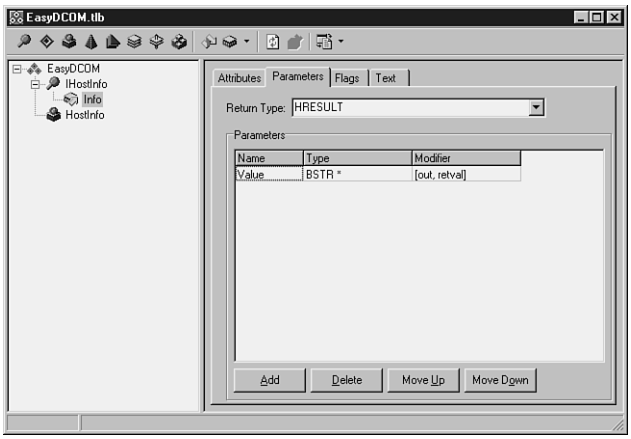


Рис. 17.10. Окно редактора библиотеки типов, в котором представлены параметры свойства `Info`

18. Щелкните на кнопке Refresh на панели инструментов редактора — C++Builder обновит файлы приложения в соответствии с внесенными изменениями.
19. Откройте в окне редактора программного кода файл HostImpl.cpp и включите в него текст метода get_Info():

```
STDMETHODIMP THostInfoImpl::get_Info(BSTR* Value)
{
    try {
        char lpBuffer[MAX_COMPUTERNAME_LENGTH + 1];
        unsigned long nSize = sizeof(lpBuffer);

        if (GetComputerName(lpBuffer, &nSize) == 0)
            return HRESULT_FROM_WIN32(GetLastError());

        WideString strVal = AnsiString().
            sprintf("Date and time at %s is: %s",
            lpBuffer, DateTimeToStr(Now()));

        *Value = strVal.Detach();
}
    catch(Exception &e) {
        return Error(e.Message.c_str(), IID_IHostInfo);
    }
    return S_OK;
} ;
```

20. Скомпилируйте и запустите на выполнение серверное приложение (нажмите клавишу <F9>).

Рассмотрим программный код извлечения свойства Info созданного сервера EasyDCOM.

Сначала вызывается API-функция Win32 GetComputerName(), помещающая в буфер lpBuffer сетевое имя компьютера, на котором выполняется это приложение. Если имя извлечь не удалось, функция возвращает код ошибки типа HRESULT.

Затем создается объект класса WideString, в который помещается имя компьютера, текущая дата и время.

Далее вызывается метод Detach() объекта класса WideString, который отсоединяет указатель BSTR от владельца — серверного приложения, поскольку модель COM требует, чтобы память для выходного параметра выделялась объектом сервера, а освобождалась объектом клиента.

Создание клиентского приложения

Комплект файлов клиентского приложения вы можете скопировать из папки EasyDCOM\Client на прилагаемом компакт-диске. Но мы покажем, как создать такое приложение самостоятельно.

Выполните следующие операции.

1. Создайте заготовку нового приложения и сохраните файл проекта в папке Client под именем EasyDCOMClient.bpr, а файл Unit1.cpp переименуйте в MainUnit.cpp.

2. Скомпонуйте главную экранную форму приложения таким образом, чтобы она выглядела как на рис. 17.11. Переименуйте объекты элементов управления в экранной форме следующим образом.

Имя, предлагаемое C++Builder по умолчанию	Новое имя
Form1	frmMain
GroupBox1	gbxLocal
GroupBox2	gbxRemote
Edit1	txtLocal
Edit2	txtRemote
Button1	cmdInfo
Button2	cmdFinish

3. Откройте файл EasyDCOMClient.cpp и добавьте в него оператор:

```
USEUNIT("..\ Server\ EasyDCOM_TLB.cpp");
```

4. В файл заголовка MainUnit.h добавьте директиву:

```
#include "..\ Server\ EasyDCOM_TLB.h"
```

5. Мы будем использовать технологию EzCOM, средства поддержки которой имеются в составе C++Builder. Поэтому объявим переменную smart-интерфейса сервера EasyDCOM.

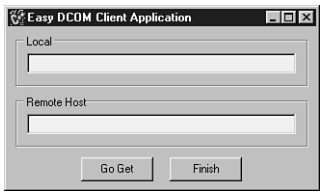


Рис. 17.11. Экранная форма клиентского приложения EasyDCOMClient

Добавьте в объявление класса TfrmMain в раздел private следующее объявление нового члена:

```
TCOMIHostInfo m_objHost;
```

6. В файле реализации экранной формы добавьте в обработчик события OnCreate следующий программный код:

```
char lpBuffer[MAX_COMPUTERNAME_LENGTH + 1];
unsigned long nSize = sizeof(lpBuffer);

GetComputerName(lpBuffer, &nSize);

txtLocal->Text = AnsiString().sprintf(
                  "Date and time at %s is: %s",
                  lpBuffer, DateTimeToStr(Now()));
```

7. EzCOM также предоставляет в распоряжение разработчика специальный класс-генератор, который имеет метод создания локального и удаленного экземпляров объекта сервера.

Дважды щелкните на кнопке cmdInfo в окне редактора экранной формы и добавьте следующий программный код в обработчик события OnClick этой кнопки:

```
if (!m_objHost.IsBound()) {
    AnsiString strHost;

    if (InputQuery("Create Server", "Enter computer name:",
        strHost)) {
```

```
        if (strHost.IsEmpty())
            OleCheck(CoHostInfo::Create(m_objHost));
        else
            OleCheck(CoHostInfo::CreateRemote(
                            WideString(strHost),
                            m_objHost));

        WideString strValue;
        OleCheck(m_objHost.get_Info(&strValue));

        txtRemote->Text = strValue;
    }
}
```

8. Дважды щелкните на кнопке `cmdFinish` в окне редактора экранной формы и добавьте в обработчик события `OnClick` этой кнопки оператор:

   ```
   Close();
   ```

9. Щелкните на заголовке экранной формы и вызовите на экран окно Object Inspector (нажмите клавишу <F11>). Откройте в этом окне вкладку Events и дважды щелкните на элементе `OnClose`, затем введите в обработчик этого события оператор:

   ```
   m_objHost.Unbind();
   ```

10. После того как вы нажмете клавишу <F9>, приложение будет скомпилировано и запущено на выполнение. Щелкните на кнопке Go Get и на экране появится диалоговое окно Create Server. Ничего не вводите в поле Machine Name, а сразу щелкните на кнопке OK — сервер будет создан на локальном компьютере.

Теперь рассмотрим подробнее программный код клиентского приложения.

В методе `OnCreate` главной экранной формы приложения извлекается и выводится на экран системное имя локального компьютера (т.е. того компьютера, на котором выполняется приложение), дата и время.

В программе обслуживания кнопки `cmdInfo` вызывается метод `IsBound()` объекта smart-интерфейса, который проверяет, создан ли экземпляр объекта сервера. Если экземпляр не создан, то функция `InputQuery()` получает от пользователя имя NETBIOS, имя DNS или IP-адрес удаленного компьютера и записывает ее в переменную `strHost`.

Если в `strHost` ничего не записано (пользователь ничего не ввел в окне запроса), экземпляр объекта сервера создается на локальном компьютере с помощью метода `Create()`. В противном случае вызывается метод `CreateRemote()`, который использует введенное пользователем имя компьютера или IP-адрес, хранящиеся в `strHost`, Этот метод пытается сформировать экземпляр объекта сервера на указанном пользователем компьютере. Функция `OleCheck()` проверяет возвращаемое значение типа `HRESULT` и генерирует исключительную ситуацию в том случае, если попытка создать экземпляр объекта сервера не удалась.

При вызове метода `get_Info()` ему передается адрес переменной типа `WideString`. Эта переменная заполняется информацией, сформированной сервером, а затем ее значение выводится в поле `txtRemote` экранной формы.

Перед закрытием формы вызывается метод `Unbind()` объекта smart-интерфейса, который выполняет отсоединение от сервера.

Поработав с DCOM-сервером и DCOM-клиентом в локальном режиме, когда оба приложения выполняются на одном компьютере, можно перейти к эксперименту с распределенным приложением, перенеся, например, сервер на другой компьютер в этой же

локальной сети. Скопируйте файл `EasyDCOM.exe` на локальный диск какого-либо компьютера в сети и запустите его на выполнение. В первом сеансе работы сервер зарегистрируется на этом компьютере.

На заметку

Если вы разместите сервер на компьютере, где установлена операционная система Windows 9X, то его придется запускать вручную, поскольку операционные системы этой группы не способны автоматически запускать сервер.

Настройка прав запуска и доступа

Будем считать, что оба компьютера — и тот, на котором выполняется серверное приложение, и компьютер клиента — входят в один домен сети Microsoft.

Ниже будет показано, как предоставить право запуска сервера всем пользователям этой сети. Выполните следующие операции.

1. Запустите программу `DCOMCnfg.exe` и дважды щелкните на имени сервера `Easy DCOM Type Library 1.0` в списке на вкладке Applications.
2. Откройте вкладку Security диалогового окна индивидуальной настройки параметров безопасности сервера и установите переключатель Use Custom Launch Permissions. Щелкните на кнопке Edit, а затем — на кнопке Add. В диалоговом окне Regisry Value Permissions дважды щелкните на имени встроенной учетной записи `Everyone`, а затем щелкните на кнопке OK (рис. 17.12). В результате всех этих манипуляций будет настроен режим, который позволяет любому пользователю, зарегистрированному в сети загружать и запускать на выполнение приложение сервера `EasyDCOM`.

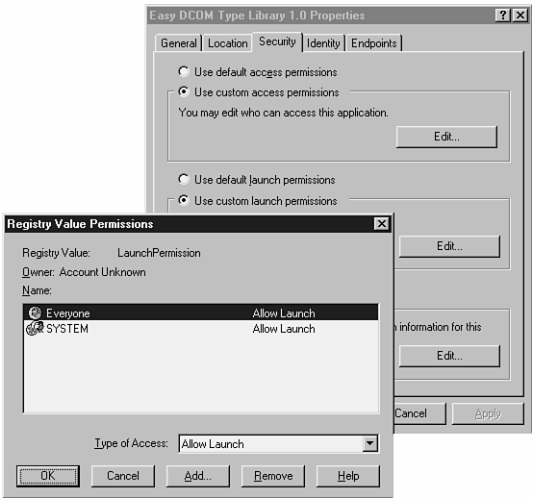


Рис. 17.12. Предоставление прав запуска сервера учетной записи `Everyone`

3. Теперь на вкладке Security установите переключатель Use Custom Access Permissions. Щелкните на кнопке Edit, а затем — на кнопке Add. В диалоговом окне Regisry Value Permissions выберите локальную или зарегистрированную в домене учетную запись, которой предоставляется право доступа к информации, формируемой сервером, а затем щелкните на кнопке OK (рис. 17.13).

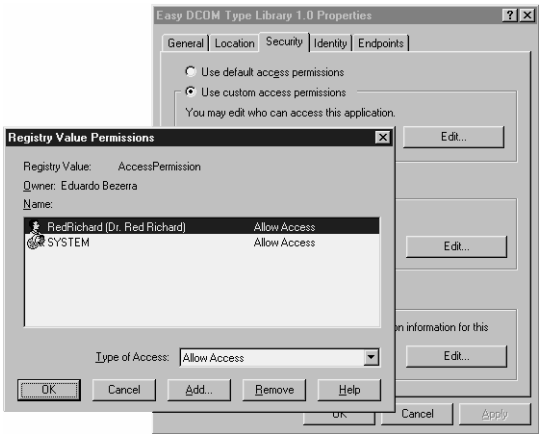


Рис. 17.13. Предоставление избранной учетной записи прав доступа к серверу

Настройка параметров персонификации сервера

Настройка параметров персонификации дает операционной системе информацию о том, от чьего имени будет выполняться серверное приложение. Как уже отмечалось, рекомендуется выбрать вариант This User и ввести данные об учетной записи, которая располагает всеми привилегиями, необходимыми для нормальной работы сервера.

На вкладке Identity (см. рис. 17.8) установите переключатель This User и введите в поле User (или отыщите в списке после щелчка на кнопке Browse) имя пользователя. В поле Password введите соответствующий пароль, а затем повторите его в поле Confirm Password. Процедура настройки завершается щелчком на кнопке OK.

Запуск распределенного приложения

Итак, подведем итоги выполненной настройке параметров безопасности нашего DCOM-приложения.

Во-первых, сервер может запускать на выполнение любой пользователь данного домена сети. Далее мы выбрали пользователя или группу пользователей, которым разрешен доступ к информации, формируемой сервером. Последний шаг — мы указали операционной системе от имени какой учетной записи (пользователя) будет работать сервер, следовательно, система будет "знать", какими правами и привилегиями обладает приложение в процессе выполнения.

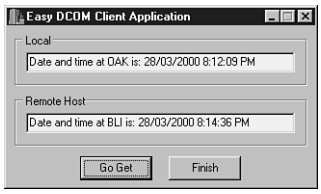


Рис. 17.14. Диалоговое окно работающего клиентского приложения EasyDCOMClient

Теперь можно опробовать наше распределенное приложение в действии.

Запустите программу `EasyDCOMClient.exe` на компьютере клиента, щелкните на кнопке Go Get и в ответ на запрос введите имя удаленного компьютера или его IP-адрес. Вы должны увидеть на экране в окне клиентского приложения нечто, похожее на то, что представлено на рис. 17.14.

Программирование средств обеспечения безопасности

В приложении, рассмотренном в предыдущем разделе, использовалась декларативная безопасность. Ниже мы рассмотрим, как реализуется в приложениях концепция *программируемой безопасности*. Возможно, вы и не подозревали, что средства поддержки COM неявно вызывают API-функции безопасности без нашего вмешательства.

Средства поддержки COM считывают настройки, зафиксированные в системном реестре, и вызывают функцию `CoInitializeSecurity()`. Функция вызывается однократно для каждого процесса, который использует механизм COM, и устанавливает соответствующий режим безопасности для всего процесса — уровень аутентификации и имена учетных записей, которым предоставлено право доступа к объекту.

Аргументы функции `CoInitializeSecurity`

Функция `CoInitializeSecurity()` является, пожалуй, самой важной среди всех API-функций, имеющих отношение к безопасности COM-объектов. Ниже приведен список аргументов вызова этой функции.

```
PSECURITY_DESCRIPTOR pVoid,   //Указатели дескрипторов
                              // безопасности
LONG cAuthSvc,                //Количество элементов в asAuthSvc
SOLE_AUTHENTICATION_SERVICE * asAuthSvc, //Массив имен реестра
void * pReserved1,    //Резервируется на будущее
DWORD dwAuthnLevel,   //Уровень аутентификации по умолчанию
DWORD dwImpLevel,     //Уровень персонификации по умолчанию
SOLE_AUTHENTICATION_LIST * pAuthList, //Данные, касающиеся
                      // аутентификации, для каждой службы
DWORD dwCapabilities, //Дополнительные возможности
                      // клиента и/или сервера
void * pReserved3     //Резервируется на будущее
```

Некоторые аргументы имеют отношение и к клиенту, и к серверу, а некоторые — только к клиенту или только к серверу.

Первый аргумент, `pVoid`, имеет отношение только к процессу, выполняющему функции сервера. Он используется для контроля прав доступа. Значением аргумента является указатель на `AppID GUID`, `SECURITY_DESCRIPTOR` или на интерфейс `IAccessControl`.

Если значением `pVoid` является **`NULL`**, средства поддержки COM не контролируют права доступа и предоставляют его всем желающим. Такое значение аргумента имеет тот же эффект, что и выбор учетной записи `Everyone` при настройке прав доступа с помощью `DCOMCnfg`. Если же значение `pVoid` отлично от **`NULL`**, то аргумент `dwAuthnLevel` (это пятый аргумент в списке) не может иметь значения **`RPC_C_AUTH_LEVEL_NONE`**.

Второй и третий аргументы, `cAuthSvc` и `asAuthSvc`, также имеют отношение только к серверу. Они используются для регистрации набора данных аутентификации с помощью COM. Элементами массива являются переменные структурного типа `SOLE_AUTHENTICATION_SERVICE`.

В состав структурного типа `SOLE_AUTHENTICATION_SERVICE` входят следующие члены:

```
typedef struct tagSOLE_AUTHENTICATION_SERVICE {
        DWORD       dwAuthnSvc;
```

```
        DWORD       dwAuthzSvc;
        OLECHAR*    pPrincipalName;
        HRESULT     hr;
}  SOLE_AUTHENTICATION_SERVICE;
```

Если значение аргумента `cAuthSvc` равно `-1`, а значение `asAuthSvc` равно `NULL`, то используется набор данных аутентификации, заданный в системе по умолчанию. NTLM (NT Lan Manager) является единственной службой аутентификации, доступной в операционной системе Windows NT 4.0, но в системе Windows 2000 можно использовать и Kerberos.

Пятый аргумент, `dwAuthnLevel`, используется для установки уровня аутентификации и касается как сервера, так и клиента. Передаваемый через этот аргумент параметр имеет тот же смысл, что и индивидуальная установка уровня аутентификации с помощью утилиты `DCOMCnfg` (см. рис. 17.5). Значение уровня аутентификации, которое передается через этот аргумент сервера, рассматривается механизмом обеспечения безопасности как минимальное. В системе будет использовано наибольшее из значений, специфицированных для сервера и клиента. Если клиент, который обращается к серверу, имеет более низкий уровень аутентификации, чем заданный для сервера, запрос отвергается.

Шестой аргумент, `dwImpLevel`, касается только клиента. Этот аргумент устанавливает уровень персонификации, который клиент должен иметь по отношению к серверу. Передаваемый через этот аргумент параметр имеет тот же смысл, что и глобальный уровень персонификации, задаваемый с помощью утилиты `DCOMCnfg` (см. рис. 17.2). Смысл использования этого аргумента в том, что он избавляет разработчика клиентского приложения от необходимости привязки к глобальным настройкам системы безопасности компьютера. Утилита `DCOMCnfg` не позволяет индивидуально настраивать параметры безопасности клиентских приложений.

Возможные значения уровня персонификации представлены набором констант:

```
RPC_C_IMP_LEVEL_DEFAULT
RPC_C_IMP_LEVEL_ANONYMOUS
RPC_C_IMP_LEVEL_IDENTIFY
RPC_C_IMP_LEVEL_IMPERSONATE
RPC_C_IMP_LEVEL_DELEGATE
```

Эти константы эквивалентны тем, которые используются при работе с `DCOMCnfg` (см. рис. 17.2).

Восьмой аргумент, `dwCapabilities`, используется как для сервера, так и для клиента. С его помощью описываются дополнительные возможности приложений. Возможные значения перечислены ниже.

- `EOAC_NONE`. Дополнительные возможности отсутствуют.
- `EOAC_MUTUAL_AUTH`. Не используется. Взаимная аутентификация поддерживается только некоторыми службами аутентификации.
- `EOAC_SECURE_REFS`. Вызовы, которые используют подсчет ссылок, должны быть аутентифицированы, поскольку это поможет избежать ошибочного освобождения ссылки.
- `EOAC_ACCESS_CONTROL`. Должно использоваться при передаче через аргумент в качестве `pVoid` указателя на интерфейс `IAccessControl`.
- `EOAC_APPID`. Индикатор того, что аргумент `pVoid` является указателем на `AppID GUID`. Функция `CoInitializeSecurity()` отслеживает значение AppID в системном реестре Registry и считывает из него параметры безопасности.

Использование функции CoInitializeSecurity()

В этом разделе будет описано, как изменить созданное ранее распределенное DCOM-приложение и включить в него возможность программной настройки параметров безопасности с помощью функции CoInitializeSecurity(). В результате это приложение сможет работать в сети Internet.

Изменения должны привести к тому, что доступ к серверу будет открыт анонимным пользователям, аутентификация и персонификация будет заблокирована. Все это выполняется через вызов функции CoInitializeSecurity().

Как уже отмечалось, функция CoInitializeSecurity() должна вызываться в каждом процессе операционной системы однократно, непосредственно перед обращением к механизму COM. Вызов должен быть организован как на стороне сервера, так и на стороне клиента сразу же после вызова функции CoInitialize().

Выполните следующие операции.

1. Откройте файл EasyDCOM.cpp.
2. Добавьте в него следующий программный код сразу же после строки, содержащей блок try в WinMain:

```
CoInitialize(NULL);

CoInitializeSecurity(NULL,
    -1,
    NULL,
    NULL,
    RPC_C_AUTHN_LEVEL_NONE,
    RPC_C_IMP_LEVEL_ANONYMOUS,
    NULL,
    EOAC_NONE,
    NULL);
```

3. Добавьте следующий программный код перед оператором return в конце WinMain:

```
CoUninitialize();
```

4. Откройте файл EasyDCOMClient.cpp и повторите операции пп. 2 и 3. Кроме того, добавьте сразу же после директивы include vcl.h еще одну директиву:

```
#include <objbase.h>
```

Теперь разберемся в том, что мы натворили.

Сначала вызывается функция CoInitialize(), которая настраивает на подключение приложения к главному потоку STA (более подробная информация о моделях потоков COM-приложений содержится в главе 16). Далее вызывается функция CoInitializeSecurity() с аргументами, соответствующими настройкам системы безопасности, о которых говорилось выше.

Аргументу pVoid мы присвоили значение **NULL**, т.е. настроили механизм COM таким образом, что доступ к серверу открыт любому пользователю.

Аргументам cAuthSvc и asAuthSvc присвоены значения -1 и **NULL**. В результате будет использован набор данных аутентификации, заданный в системе по умолчанию.

Аргументу dwAuthnLevel присвоено значение **RPC_C_AUTH_LEVEL_NONE**, поскольку мы не собираемся проводить аутентификацию пользователей, обращающихся с запросами к серверу.

Аргументу dwImpLevel присвоено значение **RPC_C_IMP_LEVEL_ANONYMOUS**, поскольку не планируется выполнять персонификацию клиента

Аргументу dwCapabilities присвоено значение **EOAC_NONE**.

После такой настройки с измененным приложением сможет работать любой пользователь.

Обеспечение безопасности в DLL-модулях клиентских приложений

Очень часто встречается ситуация, когда клиентское приложение реализуется в виде элемента управления ActiveX, используемого внутри некоторого контейнера, например Internet Explorer или IIS и Active Server Pages. Как обеспечить безопасность в этом случае? Оказывается, это не так просто.

DLL-модуль такого клиента загружается в адресное пространство процесса контейнера и в момент, когда это происходит, средства поддержки COM уже вызвали функцию CoInitializeSecurity(), явно или неявно.

При этом не имеет значения, кто именно вызвал эту функцию — приложение контейнера или COM, основываясь на данных, хранящихся в системном реестре. Поэтому вызывать функцию CoInitializeSecurity() внутри программного кода DLL слишком поздно — все необходимые настройки безопасности COM уже выполнены.

Одно из возможных решений: сформировать промежуточный внешний сервер (EXE), который настроил бы систему безопасности и вызывал удаленный сервер от имени ActiveX-клиента. А для ActiveX-клиента этот промежуточный сервер должен выглядеть, как удаленный.

Короче говоря, нельзя управлять настройкой безопасности, вызывая CoInitializeSecurity() из DLL-модуля. Это всегда следует иметь в виду при разработке клиентского DLL-модуля, который должен работать с удаленным сервером.

Реализация программируемого контроля доступа

При разработке удаленного сервера можно запрограммировать в явном виде, каким пользователям или группам пользователей будет разрешен или запрещен доступ к серверу. Эта настройка выполняется посредством аргумента pVoid функции CoInitializeSecurity(). Учтите, что при этом уровень аутентификации должен быть не ниже RPC_C_AUTH_LEVEL_CONNECT; в противном случае вызов CoInitializeSecurity() не приведет к желаемому результату.

Ниже будет показано, как при вызове CoInitializeSecurity() использовать структурный тип SECURITY_DESCRIPTOR вместо AppID GUID или интерфейса IAccessControl для передачи данных о безопасности через аргумент pVoid.

Структурный тип SECURITY_DESCRIPTOR определен следующим образом:

```
typedef struct _SECURITY_DESCRIPTOR {
   BYTE  Revision;
   BYTE  Sbz1;
   SECURITY_DESCRIPTOR_CONTROL Control;
   PSID Owner;
   PSID Group;
   PACL Sacl;
   PACL Dacl;
}  SECURITY_DESCRIPTOR;
```

Формирование дескриптора безопасности с помощью средств Win32 Security API — это скорее колдовство, чем искусство. Но C++Builder 5 и имеющаяся в его составе библиотека ATL значительно упрощают эту задачу, предоставляя в распоряжение программиста класс CSecurityDescriptor.

На заметку

Программное изменение списков управления доступом (Access Control Lists — ACL) с помощью Win32 Security API — процедура довольно сложная, требующая тщательно продуманной программы и длительного тестирования.

В дополнительном пакете Service Pack 2 к операционной системе Windows NT 4.0 имеется специальный интерфейс `IAccessControl`, который реализован в системном классе `CLSID_DCOMAccessControl`. Этот класс также можно использовать при программировании внесения изменений в списки управления доступом в качестве альтернативы обращения к "чистым" функциям Win32 Security API.

Класс `CSecurityDescriptor` можно использовать в любых ситуациях, когда возникает необходимость в формировании переменной структурного типа `SECURITY_DESCRIPTOR`, поскольку в его составе имеются операторы конвертирования. В помощью методов этого класса можно инициализировать новый дескриптор безопасности, который определяет права доступа конкретной учетной записи.

В список управления доступом обязательно должна быть включена встроенная системная учетная запись `SYSTEM`, поскольку программа управления службами (Service Control Manager — SCM) использует эту учетную запись для обслуживания COM-серверов.

Чтобы продемонстрировать, как организуется управление правами доступа определенных учетных записей к DCOM-серверу, изменим созданную ранее программу `EasyDCOM` таким образом, что сервер будет принимать запросы только от клиента, зарегистрированного в учетной записи `Guest`. При установке операционной системы эта учетная запись по умолчанию заблокирована.

1. Откройте файл `EasyDCOM.cpp` и добавьте следующий программный код сразу же после директивы `include atlmod.h`:

```
#include <atl\ atlcom.h>
```

2. Перед оператором вызова функции `CoInitialize()` добавьте следующий программный код:

```
CSecurityDescriptor sd;
sd.InitializeFromProcessToken();
sd.Allow("Guest", COM_RIGHTS_EXECUTE);
sd.Allow("NT_AUTHORITY\ \ SYSTEM", COM_RIGHTS_EXECUTE);
```

3. Измените оператор вызова `CoInitializeSecurity()`:

```
CoInitializeSecurity(sd, -1, NULL, NULL,
                     RPC_C_AUTHN_LEVEL_CONNECT,
                     RPC_C_IMP_LEVEL_IDENTIFY,
                     NULL, EOAC_NONE, NULL);
```

4. Откройте файл `EasyDCOMClient.cpp` и удалите строки, в которых вызываются функции `CoInitialize()`, `CoInitializeSecurity()` и `CoUninitialize()`.

Для проведения экспериментов с новым вариантом приложения нам придется разблокировать учетную запись `Guest` и снабдить эту учетную запись одним и тем же паролем на компьютерах сервера и клиента. Альтернативный вариант — зарегистрироваться как `Guest` на хост-компьютере сервера и запустить клиентское приложение с этого компьютера.

В принципе имя учетной записи `Guest` можно заменить именем любой другой локальной учетной записи или учетной записи домена. Для этого при передаче имени учетной записи методу `Allow()` объекта класса `CSecurityDescriptor` нужно использовать формат <*DOMAIN*>\<*UserOrGroup*> или <*MACHINE*>\<*UserOrGroup*>. Если не передать имя домена

<*DOMAIN*> или компьютера <*MACHINE*>, будет подразумеваться имя того компьютера, на котором выполняется программа.

Если попробовать запустить измененное приложение не зарегистрировавшись перед этим в учетной записи Guest, то от сервера поступит сообщение об ошибке Access Denied (доступ запрещен). Это происходит потому, что в новой версии приложения программно разрешен доступ к серверу только для учетной записи Guest.

На заметку

В составе операционной системы Windows 2000 имеется утилита RunAs. В ее помощью можно запускать на выполнение любое приложение "от имени" указанной учетной записи. При отладке серверных приложений можно с помощью этой утилиты назначить любую учетную запись, от имени которой выполняется клиентское приложение.

Рассмотрим подробнее внесенные в проект изменения.

Сначала инициализируется объект sd класса CSecurityDescriptor, который инкапсулирует новый дескриптор безопасности.

Далее вызывается метод InitializeFromProcessToken() объекта, который выполняет инициализацию членов структурного типа дескриптора.

Методу Allow() передаются в качестве аргументов имя учетной записи Guest и значение COM_RIGHTS_EXECUTE, которое определяет права доступа для этой учетной записи. Аналогично в список доступа включается и встроенная системная учетная запись System, которая фигурирует в списке аргументов как NT_AUTHORITY\SYSTEM.

Последний этап — вызов функции CoInitializeSecurity(). Этой функции в качестве первого аргумента передается объект sd класса CSecurityDescriptor. Значение RPC_C_AUTHN_LEVEL_CONNECT задает уровень аутентификации, а значение RPC_C_IMP_LEVEL_IDENTIFY — уровень персонификации. Для всех других параметров системы безопасности задаются значения по умолчанию.

Реализация системы безопасности на уровне отдельного интерфейса

До сих пор мы рассматривали такой способ организации системы безопасности, который предполагает защиту на уровне всего процесса. Этот вариант системы защиты настраивается с помощью функции CoInitializeSecurity(). Но иногда необходимо организовать систему защиты на уровне вызова отдельных методов сервера, т.е. предоставить разным категориям пользователей разные права в отношении тех или иных методов сервера.

Интерфейс IClientSecurity

Средства поддержки COM помогают программисту организовать дисциплину защиты на уровне отдельных интерфейсов как для клиентских приложений, так и для серверных. На стороне клиента имеется базовый уровень режима удаленного доступа, представленный Proxy Manager. Этот уровень реализует интерфейс IClientSecurity. На стороне сервера аналогичную функцию выполняет Stub Manager, который реализует интерфейс IServerSecurity.

В число методов интерфейса IClientSecurity входят QueryBlanket(), SetBlanket() и CopyProxy().

Метод QueryBlanket() извлекает данные аутентификации, которые используются proxy-интерфейсом. Эти данные передаются функции CoInitializeSecurity() при инициализации объекта DCOM-клиента. Если параметр безопасности не имеет значения для приложения, то соответствующий аргумент при вызове QueryBlanket() должен иметь значение NULL.

Метод SetBlanket() устанавливает значения параметров аутентификации, приложимые к определенному proxy-интерфейсу. Созданный набор параметров распространяется на всех клиентов этого proxy-интерфейса.

Метод CopyProxy() создает копию proxy-интерфейса, которую можно использовать в дальнейшем в методе SetBlanket(). Этот метод позволяет клиентам независимо друг от друга менять настройки системы безопасности своих интерфейсов.

Список аргументов методов QueryBlanket() и SetBlanket() повторяет список аргументов функции CoInitializeSecurity(), за исключением седьмого аргумента — pAuthInfo. Этот аргумент зависит от используемого провайдера службы безопасности. Если используется провайдер NTLM, то этот аргумент является указателем на структурный тип COAUTHIDENTITY.

Определение структурного типа COAUTHIDENTITY имеет следующий вид:

```
typedef struct _COAUTHIDENTITY
{
    USHORT *User;
    ULONG UserLength;
    USHORT *Domain;
    ULONG DomainLength;
    USHORT *Password;
    ULONG PasswordLength;
    ULONG Flags;
}  COAUTHIDENTITY;
```

Заполнив поля этой структуры соответствующими значениями — именем пользователя, его паролем и информацией о домене, — можно через аргумент pAuthInfo передать указатель на эту структуру и таким образом использовать для всех вызовов параметры безопасности выбранного вами пользователя. Если в качестве значения аргумента pAuthInfo передать NULL, то при каждом вызове COM-метода система безопасности будет анализировать права, которыми располагает процесс клиента.

В составе COM имеются функции оболочки CoQueryProxyBlanket(), CoSetProxyBlanket() и CoCopyProxy() для соответствующих методов интерфейса IClientSecurity.

Интерфейс IServerSecurity

На стороне сервера аналогичная задача решается с помощью интерфейса IServerSecurity, Методы этого интерфейса позволяют выполнить идентификацию клиента и персонификацию его прав. Внутри вызова метода нужно вызвать функции CoGetCallContext() и получить с ее помощью указатель на интерфейс IServerSecurity. В перечень методов IServerSecurity входят QueryBlanket(), ImpersonateClient(), RevertToSelf() и IsImpersonating().

Метод QueryBlanket() выполняет ту же работу, что и его “собрат” в интерфейсе IClientSecurity. Этот метод возвращает текущие параметры безопасности. Например, с помощью метода QueryBlanket() можно выяснить, использует ли клиент шифрование, или получить информацию, идентифицирующую текущего клиента.

В программу обработки запроса клиента к определенному методу сервера можно вставить вызов ImpersonateClient() и предоставить таким образом серверу возможность работать “от имени” текущего клиента. Уровень персонификации текущего клиента будет в дальнейшем определять, может ли сервер в процессе обработки запроса получить доступ к определенным системным ресурсам.

Метод RevertToSelf() восстанавливает первоначальные параметры безопасности, назначенные серверу, и прекращает использование севером параметров безопасности клиента. Учти-

те, что вызов RevertToSelf() необязательно должен следовать после ImpersonateClient() — средства поддержки COM автоматически восстанавливают "родные" параметры безопасности сервера по завершении обработки программой сервера каждого метода.

Метод IsImpersonating() используется для проверки, от чьего имени действует в текущий момент сервер — от имени клиента или от своего собственного.

Средства поддержки COM содержат функции оболочки CoQueryClientBlanket(), CoImpersonateClient() и CoRevertToSelf() для перечисленных методов интерфейса IServerSecurity.

Приложение Blanket

Для того чтобы показать, как на практике используются интерфейсы IServerSecurity и IClientSecurity, создадим совершенно новое приложение. В этом приложении можно будет менять параметры идентификации клиента, а сервер будет извлекать текущие параметры текущего клиента при поступлении запроса от него. Сервер будет формировать локальный файл, используя параметры безопасности текущего клиента. Потом с помощью Windows Explorer можно будет просмотреть свойства сформированного файла и выяснить, кого же операционная система посчитала его владельцем.

Прежде чем начинать эксперименты с этим приложением, я сформировал две локальных учетных записи: CommonUser и ExtraUser. Обе являются членами группы User как на компьютере сервера, так и на компьютере клиента. Затем я зарегистрировался на компьютере клиента как пользователь с учетной записью ExtraUser.

Для настройки параметров безопасности сервера я воспользовался утилитой DCOMCnfg — разрешил доступ к серверу и его запуск всем пользователям и установил, что сервер будет пользоваться правами доступа учетной записи Administrator (см. рис. 17.8).

Ниже будет описан процесс создания этого серверного приложения, но полный комплект его файлов можно скопировать и с прилагаемого компакт-диска. Файлы находятся в папке Blanket\ Server.

На заметку При отсутствии домена NT Server компьютер рабочей станции является фактически единственным членом собственного домена.

Для создания нового серверного приложения выполните следующие операции.

1. Сформируйте EXE-приложение сервера Blanket с объектом автоматизации и сопряженным классом ObjBlanket. Далее с помощью редактора библиотеки типов TLE добавьте в него два метода:

```
[id(1)] HRESULT _stdcall BlanketInfo([out, retval] BSTR * Value);
[id(2)] HRESULT _stdcall CreateFile([in] BSTR Value);
```

2. Теперь нужно реализовать эти методы. Откройте файл ObjBlanketImpl.cpp и добавьте в его метод BlanketInfo следующий программный код:

```
try {
    *Value = NULL;
    LPWSTR pPrivs;

    OleCheck(CoQueryClientBlanket(NULL, NULL, NULL, NULL, NULL,
                                  (LPVOID*)&pPrivs, NULL));
```

```
    WideString strInfo = pPrivs;
    *Value = strInfo.Detach();
}
catch(Exception &e) {
    return Error(e.Message.c_str(), IID_IObjBlanket);
}
return S_OK;
```

В этом методе функция `CoQueryClientBlanket()` используется для извлечения имени владельца клиента, пославшего запрос, а затем это имя возвращается вызывающей программе. При обращении к функции всем аргументам присваивается значение **NULL**, кроме шестого, через который передается указатель `pPrivs`. Тем самым мы даем знать функции `CoQueryClientBlanket()`, что нас интересует только идентификационные данные клиента.

3. В метод `CreateFile()` добавьте следующий программный код:

```
try {
    OleCheck(CoImpersonateClient());

    HANDLE hFile = ::CreateFile(AnsiString(Value).c_str(),
                                GENERIC_WRITE,
                                FILE_SHARE_WRITE,
                                NULL,
                                CREATE_ALWAYS,
                                FILE_ATTRIBUTE_NORMAL,
                                NULL);

    if (INVALID_HANDLE_VALUE == hFile)
        return HRESULT_FROM_WIN32(GetLastError());

    CloseHandle(hFile);
    OleCheck(CoRevertToSelf());
}
catch(Exception &e) {
    return Error(e.Message.c_str(),IID_IObjBlanket);
}
return S_OK;
```

При выполнении этого метода сначала вызывается функция `CoImpersonateClient()` и начинается персонификация клиента. Нам необходима информация о клиенте, поскольку наша цель — сформировать локальный файл, используя права клиента, пославшего запрос, а не права "владельца" сервера (в данном случае мы собираемся сделать владельцем процесса сервера учетную запись `Administrator`). Таким образом, если обращение к функции даст ожидаемый результат, мы увидим это, заглянув в свойства созданного файла, — владельцем файла должна стать учетная запись, от имени которой работает клиент.

Прежде чем выполнение метода `CreateFile()` завершится, файл закрывается и вызывается функция `CoRevertToSelf()`, которая возвращает серверу "законного" владельца.

Теперь можно скомпилировать проект. Сформированный выполняемый файл перенесите на другой компьютер в сети и запустите его, чтобы приложение самостоятельно зарегистрировалось в системном реестре.

Комплект файлов клиентского приложения находится на прилагаемом к книге компакт-диске в папке `Blanket\ Client`.

Клиентское приложение также формируется в виде выполняемого (EXE) файла. Для создания этого приложения выполните следующие операции.

1. Создайте новый проект в среде C++Builder и сохраните его под именем `BlanketClient`, а файл `Unit1.cpp` переименуйте в `MainUnit.cpp`.
2. Скомпонуйте экранную форму клиентского приложения. которая должна выглядеть, как показано на рис. 17.15. Элементам управления в экранной форме присвойте новые наименования взамен тех, которые C++Builder предлагает по умолчанию.

Имя, предлагаемое C++Builder по умолчанию	Новое имя
Form1	frmMain
RadioButton1	rdoLoged
RadioButton2	rdoIdnty
GroupBox1	gbxIdn
GroupBox1	gbxInfo
Label1	lblUser
Label2	lblDomain
Label3	lblPwrd
Edit1	txtUser
Edit2	txtDomain
Edit3	txtPwrd
Edit4	txtInfo
Button1	cmdCallServer
Button2	cmdSetIdentity
Button3	cmdGetInfo
Button4	cmdCreateFile
Button5	cmdFinish

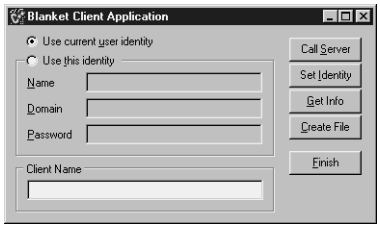


Рис. 17.15. Экранная форма клиентского приложения `BlanketClient`

3. Откройте файл `MainUnit.h` и добавьте новую директиву включения:

```
#include "..\ Server\ Blanket_TLB.h"
```

4. Добавьте в объявление производного класса `TfrmMain` два защищенных (`private`) члена:

```
TCOMIObjBlanket m_objBlanket;
SEC_WINNT_AUTH_IDENTITY m_pAuthInfo;
```

5. В объявление этого же класса добавьте два защищенных (`private`) метода:

```
void GetAuthInfo();
void SetAuthInfo();
```

Полный программный код из файла `MainUnit.cpp` представлен в листинге 17.1.

Листинг 17.1. Файл `MainUnit.cpp`, реализация главной экранной формы приложения `BlanketClient`

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

#pragma package(smart_init)
#pragma resource "*.dfm"

TfrmMain *frmMain;

__fastcall TfrmMain::TfrmMain(TComponent* Owner) : TForm(Owner)
{
    m_AuthInfo.Flags    = SEC_WINNT_AUTH_IDENTITY_ANSI;
    m_AuthInfo.User     = NULL;
    m_AuthInfo.Domain   = NULL;
    m_AuthInfo.Password = NULL;
}

void __fastcall TfrmMain::FrmMain_Close(TObject *Sender,
       TCloseAction &Action)
{
    m_objBlanket.Unbind();

    delete[] m_AuthInfo.User;
    delete[] m_AuthInfo.Domain;
    delete[] m_AuthInfo.Password;
}

void __fastcall TfrmMain::CmdCallServer_Click(TObject *Sender)
{
    if (!m_objBlanket.IsBound()) {
        AnsiString strHost;

        if (InputQuery("Create Server",
               "Enter computer name:", strHost))
        {
            if (strHost.IsEmpty())
                OleCheck(CoObjBlanket::Create(m_objBlanket));
            else
                OleCheck(CoObjBlanket::CreateRemote(
                       WideString(strHost),
                       m_objBlanket));
            ShowMessage("Server created");
        }
    }
}

void __fastcall TfrmMain::CmdCreateFile_Click(TObject *Sender)
{
```

```
    if (m_objBlanket.IsBound()) {
        AnsiString strFile;

        if (InputQuery("Create File",
                    "Enter path and file name:", strFile)) {
            OleCheck(m_objBlanket.CreateFile(
                                  WideString(strFile)));
            ShowMessage("File created");
        }
    }
}

void __fastcall TfrmMain::CmdFinish_Click(TObject *Sender)
{
    Close();
}

void __fastcall TfrmMain::CmdGetInfo_Click(TObject *Sender)
{
    if (m_objBlanket.IsBound())
        GetAuthInfo();
}

void __fastcall TfrmMain::CmdIdentity_Click(TObject *Sender)
{
    gbxIdn->Enabled = rdoIdnty->Checked ? true : false;

    txtDomain->Color = rdoIdnty->Checked ? clWindow : clBtnFace;
    txtPwrd  ->Color = rdoIdnty->Checked ? clWindow : clBtnFace;
    txtUser  ->Color = rdoIdnty->Checked ? clWindow : clBtnFace;
}

void __fastcall TfrmMain::CmdSetIdentity_Click(TObject *Sender)
{
    if (m_objBlanket.IsBound()) {
        if (rdoIdnty->Checked)
            SetAuthInfo();

        OleCheck(CoSetProxyBlanket(m_objBlanket,
                                   RPC_C_AUTHN_WINNT,
                                   RPC_C_AUTHZ_NONE,
                                   NULL,
                                   RPC_C_AUTHN_LEVEL_CONNECT,
                                   RPC_C_IMP_LEVEL_IMPERSONATE,
                          rdoIdnty->Checked ? &m_AuthInfo : NULL,
                                   EOAC_NONE));
    }
}

void TfrmMain::GetAuthInfo()
```

```
{
    WideString strInfo;
    OleCheck(m_objBlanket.BlanketInfo (&strInfo));
    txtInfo->Text = strInfo;
}

void TfrmMain::SetAuthInfo()
{
    delete[] m_AuthInfo.User;
    delete[] m_AuthInfo.Domain;
    delete[] m_AuthInfo.Password;

    m_AuthInfo.UserLength     = txtUser  ->Text.Length();
    m_AuthInfo.DomainLength   = txtDomain->Text.Length();
    m_AuthInfo.PasswordLength = txtPwrd  ->Text.Length();

    m_AuthInfo.User     = (PUCHAR)new
                          TCHAR[txtUser  ->Text.Length()+1];
    m_AuthInfo.Domain   = (PUCHAR)new
                          TCHAR[txtDomain->Text.Length()+1];
    m_AuthInfo.Password = (PUCHAR)new
                          TCHAR[txtPwrd  ->Text.Length()+1];

    lstrcpy((LPTSTR)m_AuthInfo.User,
                       (LPCTSTR)txtUser  ->Text.c_str());
    lstrcpy((LPTSTR)m_AuthInfo.Domain,
                       (LPCTSTR)txtDomain->Text.c_str());
    lstrcpy((LPTSTR)m_AuthInfo.Password,
                       (LPCTSTR)txtPwrd  ->Text.c_str());
}
```

В конструкторе `TfrmMain` инициализируются некоторые члены объекта `m_AuthInfo`. Средства поддержки COM отслеживают идентификационные данные, содержащиеся в этой структуре.

Защищенный метод `SetAuthInfo()` выполняет вспомогательную работу при операциях со структурным типом `SEC_WINNT_AUTH_IDENTITY`. Он помогает избежать утечки памяти и заполняет информацией члены этого структурного типа.

Обработчик события `Click` кнопки Set Identity изменяет пользователя клиентского приложения. Для этого вызывается функция `CoSetProxyBlanket()`, которая передает Proxy Manager информацию из объекта `m_AuthInfo`.

Можно переключиться либо на учетную запись текущего пользователя (аргументу `pAuthInfo` присваивается значение **`NULL`**), либо на учетную запись, параметры которой хранятся в `m_AuthInfo`.

При вызове `CoSetProxyBlanket()` устанавливается уровень персонификации `RPC_C_IMP_LEVEL_IMPERSONATE`, который позволяет серверу работать от имени клиента, используя параметры доступа, заданные в аргументе `pAuthInfo`.

Обработчик события кнопки Get Info вызывает метод `BlanketInfo()` сервера и получает от него данные "обратной связи" — имя текущего пользователя. Результат помещается в поле `txtInfo`.

Обработчик события кнопки Create File формирует новый файл на удаленном компьютере сервера. Этот обработчик запрашивает у пользователя путь и имя файла и передает эти данные методу `CreateFile()` сервера. Затем сервер создает новый файл, используя параметры пользователя, которые установлены в клиентском приложении.

Теперь наступило время опробовать созданное приложение на практике.

1. Запустите программу `BlanketClient.exe` и щелкните на кнопке Call Server. Введите имя компьютера сервера — будет создан экземпляр сервера `Blanket`.
2. Щелкните на кнопке Get Info. После этого экранная форма клиентского приложения должна выглядеть примерно так, как на рис. 17.16. Это — результат работы обработчика события кнопки Get Info в режиме, когда клиентское приложение использует права текущего зарегистрированного пользователя `ExtraUser`.
3. Щелкните на кнопке Create File и в ответ на запрос введите параметры формируемого файла **`C:\ RemFile1.Txt`** — сервер сформирует файл `RemFile1.Txt` в корневом каталоге диска `C:` компьютера, на котором выполняется серверное приложение.

   На рис. 17.17 показано окно свойств этого файла. В этом окне отчетливо видно, что файл создан от имени пользователя `ExtraUser`.

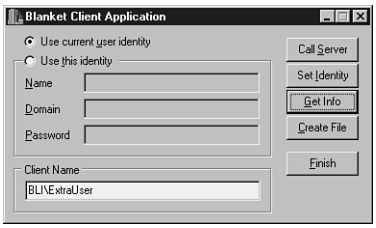


Рис. 17.16. Окно клиентского приложения `BlanketClient` в режиме использования прав текущего зарегистрированного пользователя

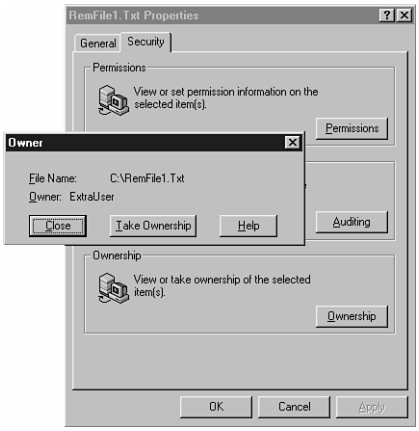


Рис. 17.17. В окне свойств файла `RemFile1.Txt` видно, что его владельцем является пользователь с учетной записью `ExtraUser`

4. Теперь включите переключатель Use this identity, а в расположенные ниже поля введите параметры регистрации учетной записи `CommonUser`. Щелкните на кнопке Set Identity — тем самым вы настроите нового владельца клиентского приложения, которое будет теперь работать от имени учетной записи `CommonUser`. На рис. 17.18 показано, как после щелчка на кнопке Get Info отреагирует на такое изменение сервер при передаче ему запроса.
5. Опять щелкните на кнопке Create File. На этот раз попросите создать файл `RemFile2.Txt`.
6. На рис. 17.19 показано, как будет выглядеть окно свойств нового файла. Видно, что его владельцем операционная система сочла учетную запись `CommonUser`.

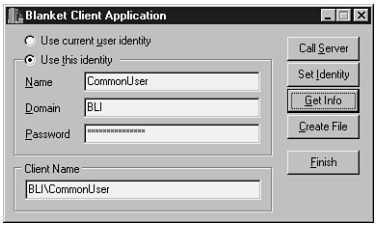


Рис. 17.18. Окно клиентского приложения `BlanketClient` в режиме использования прав учетной записи `CommonUser`

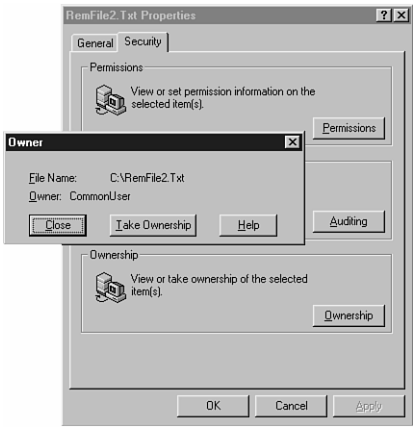


Рис. 17.19. В окне свойств файла `RemFile2.Txt` видно, что его владельцем является пользователь с учетной записью `CommonUser`

Резюме

Использование модели DCOM — довольно обширная, тема заслуживающая отдельной книги. В технологии использования DCOM имеется много нюансов, а в этой главе из-за ограниченности объема я имел возможность познакомить вас только с наиболее существенными моментами программирования распределенных приложений в рамках этой технологии. Основное же внимание было уделено обеспечению безопасного взаимодействия сервера и клиентов. Надеюсь, вы научились пользоваться утилитой `DCOMCnfg` для настройки параметров безопасности компонентов распределенных приложений. Вы также узнали, как с помощью функции `CoInitializeSecurity()` и интерфейсов `IClientSecurity` и `IServerSecurity` организовать программное управление параметрами безопасности.

Думаю, вам стало понятно, какие широкие возможности открывает DCOM перед теми, кто желает создавать приложения, выполняемые на разных компьютерах.

Тем, кто собирается серьезно заниматься созданием распределенных приложений, я настоятельно рекомендую тщательно изучить главу 19, в которой идет речь о принципах организации многоуровневых приложений с использованием MIDAS 3. Технология MIDAS является одной из альтернатив использованию DCOM в этой области программирования.

Глава

18

Еще на шаг вперед: COM+

Айонел Муньоз

С появлением на рынке операционной системы Windows 2000 возникли и новые технологии создания распределенных приложений. Специалисты из Microsoft навязывают пользователям Internet-технологию DNA (Distributed interNet Applications architecture — архитектура распределенных приложений Internet) — собственный вариант трехуровневой архитектуры приложений типа "клиент/сервер". Но это не более чем вновь изобретенное колесо, поскольку уже существует множество хорошо зарекомендовавших себя решений. Одним из них и является архитектура COM+.

Поскольку модель COM первоначально была разработана применительно к локальным программным системам, ее эволюция в направлении распределенных систем была довольно медленной. Модель DCOM была по сути временным решением — первым шагом в попытке расширить масштабы применения COM и сделать ее приемлемой для создания распределенных программных приложений. Некоторое усовершенствование DCOM было достигнуто в результате внедрения серверов Microsoft Transaction Server (MTS) и Microsoft Message Queue server (MSMQ), хотя и после этого осталось еще множество нерешенных вопросов администрирования и программирования приложений такого типа. Решительным шагом вперед в этом направлении стало появление на рынке технологии COM+.

COM+ это, возможно, первая технология, разработанная в Microsoft, которая серьезно облегчила жизнь программистам, использующим в своих разработках COM-объекты. Ее философия и политика администрирования ориентированы на корпоративные приложения, и COM+ можно со всей определенностью рассматривать как интеграцию идей DCOM, MTS и MSMQ.

Квинтэссенцией COM+ являются службы COM+ (COM+ Services). Они упрощают создание, эксплуатацию и внедрение COM-компонентов и являются существенным элементом в обеспечении масштабируемости приложений. В версии C++Builder 5 предпринята первая попытка организовать поддержку служб COM+ Services в рамках программных продуктов Borland. В состав среды разработки C++Builder 5 включены мастера, обеспечивающие формирование компонентов, которые работают с событиями и транзакциями COM+. В этой главе будет представлен обзор возможностей и технических характеристик служб COM+, причем основное внимание будет уделено именно тем из них, которые поддерживаются средой C++Builder. Итак, сделайте глубокий вдох — мы отправляемся в путь по неизведанным дорожкам в лесу, который называется COM+.

Введение в технологию COM+

Все, что вы знаете о технологии COM, в том числе и сведения, изложенные в главе 16, можно использовать при работе с моделью COM+. Но в COM+ появились и новые концепции, на которых я хочу остановить ваше внимание, прежде чем мы займемся собственно программированием. Наиболее существенными новыми понятиями в этой модели, отличающими ее от традиционной модели COM, являются *приложения COM+* (COM+ Applications), *каталог COM+* (COM+ Catalog) и *службы COM+* (COM+ Services). Описание этих понятий — тема данного раздела.

Приложения COM+

Приложение COM+ является центральным объектом системы администрирования COM+. Приложение состоит из именованных групп взаимодействующих компонентов COM+, которые упакованы вместе таким образом, что их легко конфигурировать и внедрять в пакетном режиме.

С помощью службы работы с компонентами ***Windows 2000 Component Services*** программист может создавать приложения COM+ и настраивать их характеристики — как те, что определяют взаимодействие с подсистемой безопасности операционной системы, так и атрибуты отдельных компонентов.

На заметку

Службу ***Component Services*** можно найти в Windows 2000, выбрав в меню Start⇨Settings⇨Control Panel⇨Administrative Tools⇨Component Services. Как работать с этой службой, будет рассказано в одном из последующих разделов этой главы.

Каждый COM-компонент, который добавляется в приложение, называется *конфигурируемым компонентом* (в противном случае он называется *неконфигурируемым* компонентом). Только конфигурируемые компоненты могут воспользоваться услугами служб COM+. Для того чтобы COM-компонент мог быть включен в состав приложения COM+, он должен иметь вид DLL-модуля (встроенного сервера) и распространяться месте с информацией о применяемых в нем типах — вот почему в главе 16 я все время обращал ваше внимание на важность библиотеки типов при разработке COM-компонентов.

На заметку

Напоминаю, что C++Builder обеспечивает связывание библиотеки типов с сервером на двоичном уровне. Об этом подробно рассказано в главе 16.

Существует четыре типа приложений COM+. Главным является *приложение сервера* (или *серверное приложение*). Серверное приложение выполняется в заменяющем (surrogate) процессе и поддерживает полный спектр функций служб COM+. Как программисту вам чаще всего придется иметь дело именно с серверными приложениями COM+.

Второй тип приложений — *библиотечные*. Компоненты библиотечного приложения всегда выполняются в процессе клиента, который создал эти компоненты. В этом типе приложения не поддерживается удаленный доступ и формирование очередей вызова компонентов.

Proxy-приложения представляют собой еще один тип приложений COM+. Приложения этого типа содержат только информацию, необходимую для регистрации серверного приложения, которая может быть использована для установки этого сервера в режим удаленного доступа.

Четвертый и последний тип — *прединсталлированные приложения* COM+ (*COM+ preinstalled applications*). Приложения этого типа работают напрямую с внутренними функциями COM+ и не могут быть изменены или удалены.

Каталог COM+

Вы помните, что такое системный реестр операционной системы Windows? При работе с моделью COM+ возможностей Win32 Registry не хватает для поддержки работы новых служб. Поэтому средства поддержки COM+ располагают собственной, независимой от Win32 Registry, системной базой данных — `RegDb`. Для доступа к этой базе данных используется набор системных объектов, объединенных под общим наименованием *каталог COM+* (*COM+ Catalog*).

Каталог COM+ обеспечивает формирование и обслуживание иерархической системы COM-объектов при программной обработке данных конфигурации, которые хранятся как в системном реестре Registry, так и в `RegDb`. Доступ к этой иерархии осуществляется через `COMAdminCatalog` — COM-компонент, который предоставляет в распоряжение программиста все административные функции посредством набора коллекций и методов. Программный

идентификатор этого объекта — `COMAdmin.COMAdminCatalog`. Все объекты каталога COM+ доступны как из программ, написанных на языках создания сценариев, так и из программ, разработанных на языках с четкой системой типов.

На заметку Подробное описание набора API-функций администрирования COM+ (COM+ Administration API) и иерархии каталога COM+ можно найти в справочной системе MSDN.

В операционной системе Windows 2000 клиентом каталога COM+ по умолчанию назначена служба ***Component Services***. С помощью этой службы можно выполнять все административные процедуры, не написав ни строки программного кода. В этой главе будет описана методика работы с объектами каталога с помощью Windows 2000 ***Component Services***.

Использование служб COM+

После того как COM-компоненты включены в состав приложения COM+, это приложение может обращаться к услугам служб COM+. Именно эти службы в значительной степени определяют нетривиальные возможности созданного приложения COM+. Приведенное ниже описание служб COM+ даст вам четкое представление о них.

На заметку Службы обработки событий (Event Service) и транзакций (Transactions Service) описаны подробнее других, поскольку они поддерживаются мастерами C++Builder 5. Кроме того, предполагается, что модель COM+ будет поддерживать и базы данных в оперативной памяти (In-Memory Databases), но эта функция пока еще не реализована в существующей на сегодняшний день версии операционной системы Windows 2000.

Свободно связываемые события

Схема "публикатор/подписчики" (publisher/subscriber) является на сегодняшний день одной из самых распространенных схем взаимодействия между объектам в распределенной вычислительной среде. *Публикатор* — это любой программный объект, который извещает другие объекты о произошедших в нем изменениях (обновлении внутреннего состояния, изменении подконтрольной информации и т.п.), возбуждая определенное событие. Подписчики получают это уведомление в асинхронном режиме и адекватно на него реагируют.

В модели COM+ поддерживается довольно простая схема администрирования такого процесса с помощью оформления *подписки* (*subscription*) — своего рода контракта между публикатором и подписчиком. Как только публикатор возбуждает событие, средства поддержки COM+ рассылают уведомления подписчикам.

При использовании модели обработки событий COM+ отпадает необходимость в интерфейсе `IConnectionPointContainer`. От публикатора требуется только сформировать объект события и вызвать его методы, как это делается при работе с обычным COM-объетом. Публикатор не располагает никакой информацией о своих подписчиках. Распространение информации о событии между подписчиками осуществляют службы COM+. Именно они следят за подпиской на определенные события и выборочно распространяют уведомления о событиях между соответствующими подписчиками. Подписчики, со своей стороны, должны реализовать интерфейс этого события с теми же свойствами, что и у объекта события публикатора. Поскольку публикатор и подписчики взаимодействуют не напрямую, а через средства поддержки COM+, то принято называть подобный механизм распространения событий механизмом *свободно связываемых* событий. На рис. 18.1 схематически представлен этот механизм.

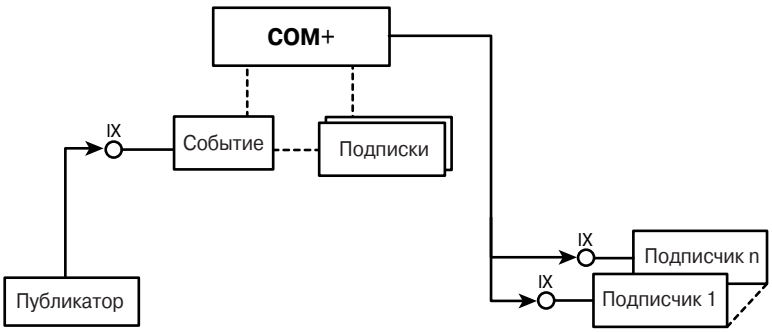


Рис. 18.1. Свободно связываемые события COM+

На заметку

В главе 16 был рассмотрен механизм распространения событий с помощью точек подключения (connection points). Основной недостаток этого механизма в том, что должна существовать непосредственная информационная связь между клиентом и сервером — клиент должен "знать" сервер, к которому он обращается, а сервер должен реализовать интерфейс `IConnectionPointContainer` для того, чтобы отослать уведомление приемнику событий клиента. Поэтому такой механизм распространения событий получил наименование механизма *жестко связанных* событий.

Набор подписок

Набор подписок — основной компонент модели обработки событий в COM+. Говорят, что подписка *постоянная* (*persistent*), если она сохраняется в каталоге COM+ и восстанавливается при перезагрузке операционной системы. Для "оформления" такой постоянной подписки обычно используется служба Windows 2000 ***Component Services***. При наличии постоянной подписки каждый раз, когда публикатор возбуждает событие, формируется объект для соответствующего подписчика. После того как подписчик завершит обработку события, объект уничтожается.

Клиент может динамически формировать *временную* (*transient*) подписку. Такая подписка не восстанавливается при перезагрузке операционной системы. Временная подписка имеет отношение к существующему объекту класса подписчика. Программист организует управление этой подпиской внутри программы клиента, напрямую обращаясь к методам каталога COM+. ***Component Services*** никакой помощи в оформлении временной подписки не оказывает.

Фильтрация событий

Средства поддержки COM+ могут также организовать *фильтрацию* событий. При формировании фильтра за основу могут браться определенные методы, а точнее, определенные в этих методах параметры. Строка определения фильтра сохраняется в подписке. Такая фильтрация получила наименование *параметрической* (*parameter filtering*).

Существуют также и *фильтры публикатора* (*publisher filters*), которые организовать несколько сложнее, но они предоставляют программисту более широкие функциональные возможности. Класс фильтра публикатора можно ассоциировать с классом события. В этом случае программист получает возможность контролировать порядок возбуждения событий и оповещения подписчиков о них.

Транзакции

Транзакции используются в распределенных приложениях для того, чтобы обеспечить совместную корректность данных (информационную целостность). Классический пример использования транзакций, кочующий из одной книги в другую, — списание денег с одного счета и запись на другой. Поскольку состояния обоих счетов в любой момент времени должны быть совместимы, необходимо процедуру перечисления денег со счета на счет выполнять как единую нераздельную операцию. Эта нераздельная операция либо выполняется полностью — т.е. выполняется и списание с одного счета и зачисление на другой, — либо полностью не выполняется — на обоих счетах сохраняются прежние суммы. Не может случиться так, что деньги с одного счета списаны, а на другой по причине какого-либо сбоя не зачислены.

Свойства транзакций

Транзакция характеризуется четырьмя свойствами, которые получили наименование набора ACID-свойств по первым буквам английских названий:

- *Неделимость* (Atomic) — транзакция является нераздельной операцией, выполняемой по принципу "все или ничего".
- *Целостность* (Consistent) — перед началом выполнения транзакции и после ее выполнения (независимо от того, зафиксирована транзакция или отвергнута) должны быть удовлетворены все ограничения, налагаемые на информационную целостность системы, в пределах которой выполняется транзакция.
- *Изолированность* (Isolated) — промежуточные состояния системы в процессе выполнения любой транзакции должны быть недоступны (скрыты) для любой другой конкурирующей транзакции.
- *Устойчивость* (Durable) — состояние, в которое переводится система в результате выполнения транзакции, должно быть устойчиво — система может находиться в этом состоянии сколь угодно долго.

Службы COM+ обеспечивают поддержку выполнения транзакций с помощью специальных *объектов транзакций COM+* (*COM+ transactional objects*). Эти объекты принимают участие в выполнении транзакций и определяют, будет она принята или отвергнута. Только после опроса всех объектов, вовлеченных в выполнение транзакции, принимается решение о том, зафиксировать ли ее как единую операцию или отвергнуть.

На заметку

"Голосование" объекта производится через интерфейс `IObjectContext`. В этой главе будет рассмотрен пример использования этого интерфейс при выполнении транзакций.

Виды объектов транзакций COM+

Ниже перечислены существующие в COM+ виды объектов транзакций.

- **Requires New Transaction** (требует новой транзакции). Объект всегда является корневым в выполнении транзакции. Даже если объект сформирован в контексте текущей транзакции, службы COM+ передают ему новую транзакцию по завершении текущей. Этот механизм используется для изолирования независимых вложенных транзакций.
- **Requires Transaction** (требует транзакцию). Объект всегда должен быть вовлечен в выполнение транзакции. Если объект создан, а транзакция не выполняется, COM+ формирует новую транзакцию и делает объект корневым для этой транзакции. В про-

тивном случае, если объект создан в контексте существующей транзакции, она распространяется на этот объект.

- **Supports Transactions** (поддерживает транзакцию). Такой объект можно охарактеризовать как "разрешает выполнение транзакции, но не принимает в этом участия". Этот атрибут может быть использован объектами, которые не нуждаются для выполнения своих задач в транзакциях, но могут взаимодействовать с объектами, которые транзакции выполняют. Если объект сформирован в контексте транзакции, он передает ее любому другому объекту двух перечисленных выше типов, которые могут быть созданы в программе.
- **Does Not Support Transactions** (не поддерживает транзакцию). Такой объект не может быть вовлечен ни в какие операции, связанные с транзакциями. Объект с таким атрибутом блокирует всякое распространение транзакции на объекты, им сформированные, даже в том случае, когда такие объекты "заинтересованы" в транзакциях.
- **Disabled** (запрещено). Такой объект блокирует автоматический механизм выполнения транзакций и берет на себя все выполняемые этим механизмом функции. Если вы достаточно ленивы (или достаточно рассудительны), то не будете использовать в своих разработках объекты подобного типа.

Контекст объекта

Данные, которые COM+ ассоциирует с каждым из зарегистрированных объектов COM+, называются *контекстом объекта*. Контекст формируется службами COM+ в тот момент, когда объект готовится к активизации и к нему можно получить доступ через интерфейс `IObjectContext`. Для инициализации контекста необходимые атрибуты выбираются из каталога COM+. Если, например, объект требует новой транзакции, таковая создается COM+ и подключается к объекту.

На заметку Компонент COM+, который включает транзакции в контекст, называется ***Distributed Transaction Coordinator*** (DTC).

Временная активизация и организация пула объектов

С помощью механизма *временной активизации* COM+ (*just-in-time* activation — *JIT*) объект транзакции активизируется в тот момент, когда к нему должен получить доступ другой объект, и после использования сразу же переводится в неактивное состояние. Активизация и прекращение активности объектов в чем-то соответствуют созданию и уничтожению объектов, но службы COM+ используют специальную функцию *Object Pooling*, которая позволяет временно сохранить неактивные объекты в памяти на случай, если они вот-вот понадобятся. Таким образом несколько повышается производительность работы приложения, поскольку можно избежать частого повторного создания одних и тех же объектов.

Для того чтобы объект транзакции можно было поместить в пул COM+, он не должен сохранять информацию о своем состоянии. Это противоречит одному из принципов объектно-ориентированного программирования, но иногда приходится приносить принципы в жертву на алтарь целесообразности и производительности. *Объект без состояния* (stateless object) — это объект, который обычно не поддерживает информацию о своем состоянии между вызовами методов. Если объект не поддерживает своего состояния между вызовами методов, COM+ может заставить объект " исчезнуть" из поля зрения между вызовами, поместив в пул, а при необходимости вновь его активизировать после извлечения из пула.

Управление ресурсами

Объекты транзакций COM+ идеально подходят для того, чтобы запрограммировать в них бизнес-правила для работы в многоуровневой среде. Управление системой в устойчивом состоянии (после завершения транзакции) осуществляется так называемыми менеджерами ресурсов (Resource Managers — RM). Обычно RM — это программный компонент, приобретаемый у независимых разработчиков, который работает с сохраняемыми данными, например Oracle или MS-SQL Server. Менеджеры ресурсов перечисляются в транзакциях COM+ и могут в прозрачном режиме вносить изменения в сохраняемое состояние системы без непосредственного обращения к объектам транзакций. Ключевым элементом, обеспечивающим работоспособность такой архитектуры, является служба распределенной координации транзакций COM+ (Distributed Transaction Coordinator — DTC), которая извещает всех менеджеров ресурсов о конечном состоянии транзакции COM+, имеющей к ним какое-либо отношение. На рис. 18.2 схематически показано, как это происходит.

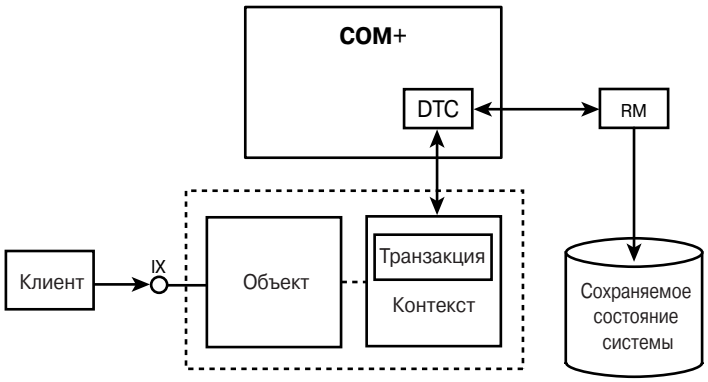


Рис. 18.2. Транзакция COM+, существующая в контексте объекта, DTC и менеджер ресурсов (RM)

Менеджеры ресурсов поступают обычно в комплекте с MTS (Microsoft Transaction Server). Разработка менеджера ресурсов — дело далеко не простое, поскольку требует организации взаимодействия с DTC на низком уровне. Однако в составе COM+ имеется специальный вид менеджера ресурсов, который именуется ***Compensated Resource Manager*** (CRM). CRM упрощают разработку менеджера ресурсов и в этой главе в одном из примеров будет продемонстрирована методика его использования в среде C++Builder.

Синхронизация

В "классической" модели COM использование *однопотоковой секционированной* модели потоков (single-threaded apartment — STA) обеспечивает прозрачность механизма синхронизации доступа к объектам, последовательно "выстраивая" вызовы в пределах одного потока. С другой стороны, многопотоковая модель (MTA) позволяет любому потоку в секции обращаться к методам объекта, но весь механизм синхронизации должен быть реализован в самом объекте.

На заметку

В главе 16 вы можете познакомиться с более подробным описанием моделей потоков и маршалинга.

Как совместить выбранную модель потоков и средства, обеспечивающие синхронизацию, — одна из основных задач, стоящих перед программистом COM-приложений. Для того чтобы помочь решить ее, в состав COM+ включен новый вид модели потоков, получивший наименование *Neutral* (нейтральная). Объект с такой моделью потока может получить вызов из любого другого потока, а это означает, что его методы выполняются в потоке вызывающего объекта. Эту модель можно рассматривать как особый вид модели MTA, но без переключения потоков приходящих вызовов. Вы спросите "А где же здесь синхронизация?". Ответ прост: в состав COM+ включен *модуль активности* (*activity unit*), который самостоятельно обеспечивает дисциплину в очереди вызовов COM-объектов (точнее, методов COM-объектов). Такой механизм синхронизации можно настраивать с помощью средств администрирования ***Component Services***. Сама по себе тема синхронизации довольно обширная и выходит за рамки этой главы, но главное, что вам следует вынести из этого раздела, состоит в следующем: при разработке компонентов COM+ используйте модель потоков *Neutral*, если у вас нет других особых соображений.

Безопасность

Средства обеспечения безопасности в составе служб COM+ в значительной мере унаследованы от операционной системы Windows NT и модели DCOM, но в COM+ требуется значительно меньше усилий для администрирования безопасности. Значительно упрощено и программирование средств безопасности в приложениях. Мы не будем здесь подробно рассматривать эти вопросы, а тех читателей, которых интересуют детали реализации безопасности приложений COM+, мы отсылаем к системе оперативной справки MSDN.

Очередь компонентов

Службы COM+ поддерживают функционирование механизма, которому разработчики присвоили наименование *Queued Component* (компонент, включаемый в очередь). Queued Component позволяет клиенту вызывать методы сервера асинхронно, причем даже тогда, когда сервер не запущен. В такой ситуации вызов помещается в очередь и дожидается, пока сервер не "оживет". Подобная "услуга" в сетевой среде оказывается очень полезной.

Балансирование загрузки системы

Сбалансировать загрузку ресурсов распределенной системы при использовании DCOM — кошмар для администратора, поскольку каждый клиент абсолютно свободен в выборе сервера для подключения. Представьте себе, что делается в локальной сети, насчитывающей около сотни компьютеров.

Для балансирования загрузки ресурсов среды в составе COM+ имеется служба ***Component Load Balancing*** (CLB). Компьютер, на котором работает эта служба, играет роль регулировщика, который динамически отдает приоритет на создание объектов тому серверу в сети, который наименее загружен в текущий момент. Алгоритмы работы CLB здесь рассматриваться не будут, но главное, что нужно запомнить из этого коротенького раздела, — CLB помогает обеспечить масштабируемость распределенной системы приложений COM+, поскольку единственным компьютером, известным всем клиентам, становится тот, на котором "работает" эта служба. Подробную информацию о CLB можно найти в справочной системе MSDN.

Программирование и использование в приложениях событий COM+

Структура программных систем, в которых используются события COM+, практически одна и та же: публикатор, который оповещает о событии в распределенной среде, объект события, в котором формируется конкретное событие, и подписчики, которые должны получить извещение о событии и адекватно на него отреагировать. В этом разделе мы продемонстрируем, как в рамках этой структуры создать с помощью C++Builder приложение агентства по найму (публикатор) и оповещать подписчиков о том, что в этом приложении появилась новая вакансия.

Создание объекта события COM+

События COM+ представляют собой классы, определение типа которых используется службами COM+ для сопоставления подписки с подписчиками. Реально экземпляры таких классов не создаются. Поэтому эти классы не имеют кода реализации, а содержат только информацию о типе. Однако сервер, который содержит такой объект, включает и программный код, необходимый для регистрации класса события. Должен с огорчением сообщить, что мастер ***COM+ Event Wizard***, который имеется в составе C++Builder 5, этот программный код не формирует.

Поэтому при создании объекта события COM+ в среде C++Builder у программиста есть два варианта действий: использовать мастер ***COM+ Event Wizard*** и самостоятельно реализовать весь недостающий программный код или, использовав мастер создания "обычных" объектов автоматизации ***Automation Object Wizard***, ввести затем в созданную им заготовку информацию о типе, которая характеризует создаваемый объект именно как объект события COM+. Мне кажется, что второй вариант предпочтительнее, поэтому в дальнейшем будет рассматриваться именно он. Будем надеяться, что в следующих версиях C++Builder разработчики учтут наши замечания и доработают мастер ***COM+ Event Wizard***.

На заметку

Я обнаружил эту проблему с помощью системы справки MSDN Windows 2000, редакции Professional. Именно этой версией я пользовался при создании примеров, рассматриваемых в этой главе.

Для создания сервера `JobEvents` нужно выполнить следующие операции (сценарий создания сервера во многом повторяет тот, что рассматривался в главе 16 при создании первого COM-сервера).

1. Запустите C++Builder.
2. Выберите в системном меню File⇨New. На экране откроется диалоговое окно New Items.
3. Выберите в этом окне вкладку ActiveX и дважды щелкните на ярлыке ActiveX Library.
4. Сохраните заготовку проекта в файле под именем `JobEvents`.
5. Еще раз откройте диалоговое окно New Items (команда меню File⇨New).
6. На все той же вкладке ActiveX дважды щелкните на ярлыке Automation Object. На экране откроется диалоговое окно New Automation Object.
7. Дайте объекту наименование `JobEvent` и, если желаете, введите его описание в поле Description. Не обращайте внимание на переключатели выбора модели потоков и флажок Generate Event Support Code. Учтите, что главное, что нам требуется, — объявление типа события и формирование программного кода регистрации. Щелкните на кнопке OK.

Мы создали простой сервер автоматизации, в котором имеется объект автоматизации с дуальным интерфейсом по умолчанию, названным IJobEvent. В этом процессе для вас ничего нового нет (если, конечно, вы уже читали главу 16). Теперь самостоятельно добавьте в интерфейс IJobEvent метод JobAvailable(). Определение метода должно выглядеть следующим образом:

```
HRESULT _stdcall JobAvailable([in ] BSTR MainCompetence );
```

На заметку Если вы забыли, как добавляется новый метод в существующий интерфейс, перечитайте главу 16.

Теперь пришло время призвать на помощь силы белой и черной магии. Выберите в левой панели диалогового окна редактора библиотеки типов TLE узел сопряженного класса JobEvent, а в правой панели — вкладку COM+. Вы увидите, что все атрибуты события заблокированы, поскольку сейчас этот сопряженный класс еще не представляет события COM+. Для того чтобы преобразовать сопряженный класс JobEvent в класс события COM+, нам не понадобится волшебный кристалл — просто следуйте за мной.

1. В левой панели окна редактора TLE выберите узел сопряженного класса JobEvent.
2. Выберите вкладку Text и отредактируйте IDL-код. Добавьте в определение сопряженного класса JobEvent атрибуты, выделенные в приведенном ниже фрагменте полужирным шрифтом. Эта новая строка определяет сопряженный класс как класс события COM+ (рис. 18.3):

```
[
uuid(FA436AEA-9F7D-4105-916C-2C9FCBCA0E6D),
version(1.0),
helpstring("JobEvent Object"),
custom(B0FC9341-5F0E-11D3-A3B9-00C04F79AD3A,-1)
]
coclass JobEvent
{
[default ] interface IJobEvent;
};.
```

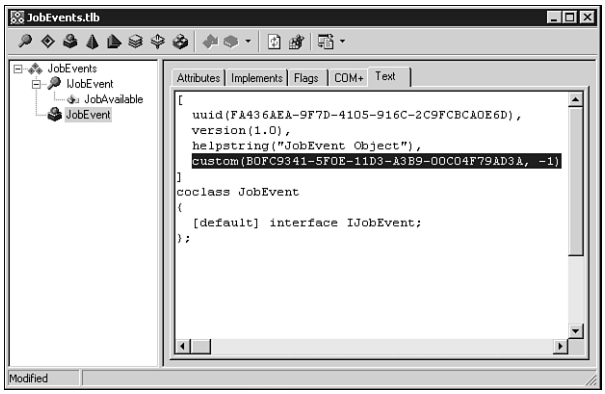


Рис. 18.3. Редактирование вручную объявления сопряженного класса JobEvent на языке IDL

Теперь опять выберите вкладку COM+ и убедитесь, что поля атрибутов события разблокированы.

Обратите внимание: мы не написали ни строки программного кода на языке C++. Для того чтобы обхитрить C++Builder, оказалось достаточно внести одно добавление в IDL-определение сопряженного класса. Сохраните результат. Теперь, после компиляции, класс события `JobEvent` размещается в модуле сервера `JobEvents.dll`, и этот класс можно зарегистрировать в приложении COM+.

Включение объекта события в приложение COM+

Сначала нам придется создать новое приложение COM+.

1. Откройте диалоговое окно Component Services. Для этого сначала выберите в меню панели задач Windows 2000 Start⇨Settings⇨Control Panel.
2. В открывшемся диалоговом окне Control Panel дважды щелкните на ярлыке Administrative Tools, а затем дважды щелкните на ярлыке Component Services — откроется диалоговое окно Component Services.
3. В левой панели этого окна выберите ветвь Console Root, а затем последовательно открывайте ветви Component Services⇨My Computer⇨COM+ Applications.
4. Выделите элемент COM+ Applications в левой панели, как показано на рис. 18.4.
5. Выберите в меню команду Action⇨New⇨Application. На экране появится первое окно мастера ***COM Application Install Wizard***.
6. Щелкните в нем на кнопке Next — появится второе окно, Install or Create a New Application (Установка или создание нового приложения) (рис. 18.5).
7. Щелкните на кнопке Create an Empty Application (Создать пустое приложение). На экране появится окно Create Empty Application.
8. В этом окне введите имя приложения **`Jobs Events`**. В зоне Activation Type (Тип активизации) установите переключатель Server Application (Серверное приложение), как показано на рис. 18.6.
9. Щелкните на кнопке Next. На экране откроется следующее окно мастера, Set Application Identity (Установка параметров идентификации приложения). Не изменяйте предлагаемое по умолчанию значение `Interactive User` (Пользователь) параметра Account (Учетная запись).

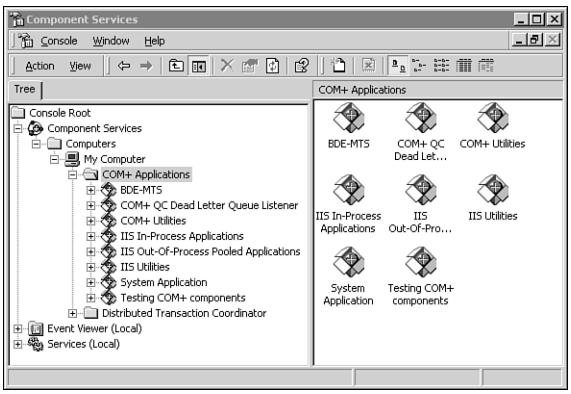


Рис. 18.4. Диалоговое окно Component Services, в котором развернута ветвь COM+ Applications

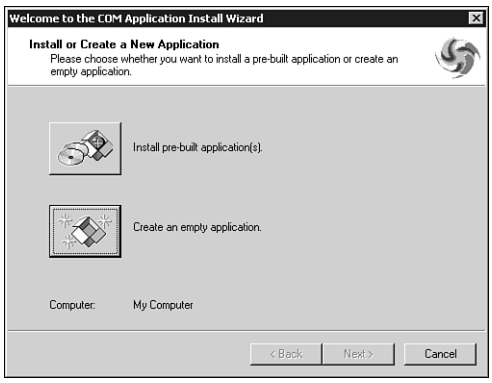


Рис. 18.5. Окно Install or Create a New Application *мастера* **COM Application Install Wizard**

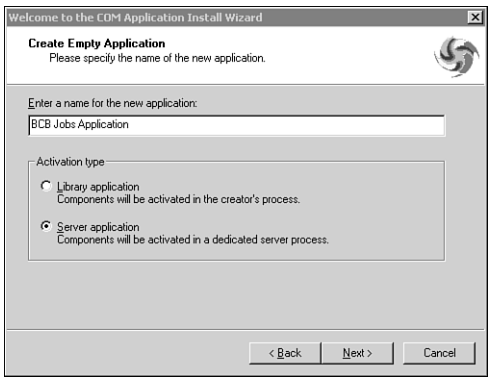


Рис. 18.6. Окно Create Empty Application *мастера* **COM Application Install Wizard**

10. Щелкните на кнопке Next. Теперь на экране откроется последнее окно мастера, который вежливо попрощается с вами.

11. Щелкните на кнопке Finish. Теперь в списке приложений COM+ в окне Component Services появится новое приложение (рис. 18.7).

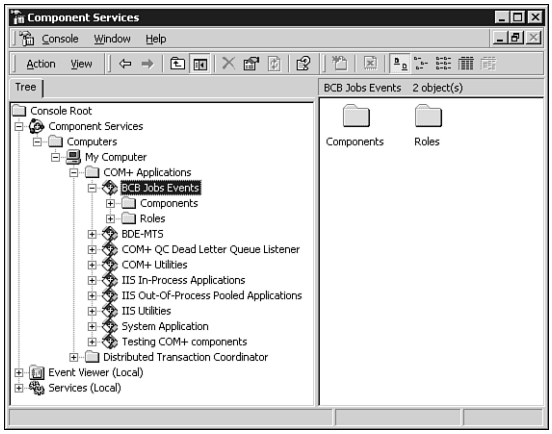


Рис. 18.7. Окно Component Services*, в котором появился элемент нового приложения* `BCB Jobs Events`

Теперь, когда мы располагаем новым приложением COM+, в него можно добавить созданный ранее класс события COM+.

1. В ветви приложения `BCB Jobs Events` выделите узел Components, а после этого в главном меню диалогового окна Component Services выберите команду Action⇨New⇨Component. Будет запущен новый мастер ***COM Component Install Wizard***, и на экране появится его первое окно.

2. Щелкните на кнопке Next. Теперь на экране появится окно Import or Install a Component мастера, показанное на рис. 18.8. Это окно позволяет выбрать режим импортирования или установки компонента.

3. Щелкните на кнопке Install New Event Class(es) (Установить новые классы событий). На экране откроется диалоговое окно, в котором нужно будет отыскать в файловой системе компьютера ранее созданный файл `JobEvents.dll`. Отыскав этот файл, дважды щелкните на его изображении. Теперь откроется диалоговое окно со списком всех классов событий выбранного сервера (рис. 18.9).

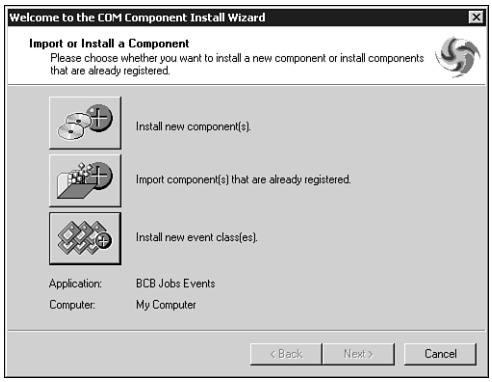


Рис. 18.8. Окно Import or Install a Component *мастера* **COM Component Install Wizard**

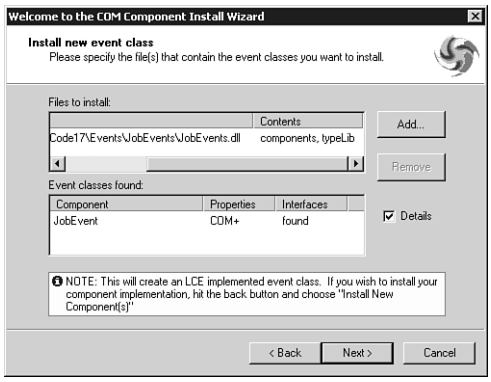


Рис. 18.9. Окно Install New Event Class *мастера* **COM Component Install Wizard**

4. Поле щелчка на кнопке Next откроется последнее окно мастера.
5. Щелкните на кнопке Finish.

Если на описанном пути вы не столкнулись в какой-либо проблемой (а их не должно быть, если вы точно выполняли все перечисленные операции), то в приложение `BCB Jobs Events` теперь будет включен новый класс событий. Следующий этап работы — создание публикатора для этого класса события.

Создание публикатора

Мы уже говорили о том, что создаваемое в среде C++Builder приложение агентства по найму на работу должно использовать сопряженный класс `JobEvent`, чтобы послать сообщение о событии (об открывшейся вакансии) `JobAvailable()` всем подписчикам. Организовать это так же просто, как и создать клиентскую программу объекта `JobEvent`. Все, что нужно сделать такой программе, — обратиться к методу `JobAvailable()` этого объекта.

Для создания публикатора, который будет оповещать своих подписчиков об открывшейся вакансии каждые пять секунд, нужно выполнить следующие операции.

1. Создайте новую заготовку приложения в среде C++Builder.
2. Переименуйте главную экранную форму приложения — пусть она называется `MainForm`. Сохраните файл модуля `Unit1.cpp` под именем `Main.cpp`, а файл проекта под именем `JobPublisher`.
3. Импортируйте в приложение библиотеку типов `JobEvents` — для этого выберите в меню C++Builder команду Project⇨Import Type. На экране откроется диалоговое окно Import Type Library.
4. В списке этого окна выберите имя библиотеки типов созданного ранее сервера событий COM+ `JobEvents Library (Version 1.0)`.

5. Щелкните на кнопке Create Unit (рис. 18.10). Теперь в состав проекта будет добавлен файл JobEvents_TLB.cpp.

6. В файл заголовка MainForm.h включите директиву #include с именем файла, содержащего информацию о типе сопряженного класса JobEvent:

```
#include "JobEvents_TLB.h "
```

7. В классе главной экранной формы объявите объект сопряженного класса JobEvent. Для этого воспользуйтесь классом оболочки TCOMIJobEvent, который объявлен в файле JobEvents_TLB.h :

```
class TMainForm :public TForm
{
   [ Пропущено... ]
   TCOMIJobEvent FJobEvent;
};
```

8. В обработчике события FormCreate() главной экранной формы организуйте создание экземпляра сопряженного класса JobEvent:

```
void __fastcall TMain-
Form::FormCreate(TObject *Sender)
{
   FJobEvent = CoJobEvent::Create();
}
```

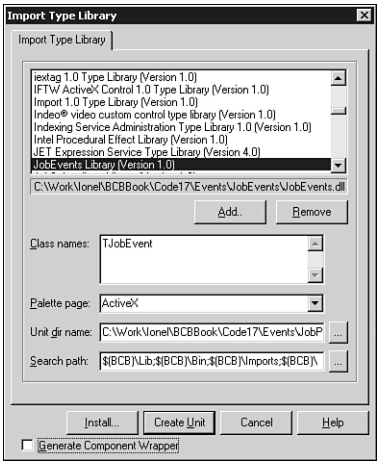


Рис. 18.10. Импортирование в приложение библиотеки типов JobEvents

9. Добавьте в состав класса экранной формы приложения компонент TTimer. Присвойте его свойству Interval значение **5000** (это означает установку интервала в 5 сек), а свойству Enabled значение **True**.

10. Присвойте объекту класса TTimer наименование **JobTimer**.

11. Создайте обработчик события OnTimer этого объекта.

12. В этом обработчике события нужно вызывать метод JobAvailable() объекта класса JobEvent и передавать ему в качестве аргумента MainCompetence величину типа WideString, значение которой выбирается случайно из массива, состоящего из пяти строк. Программный код обработчика должен выглядеть следующим образом:

```
void __fastcall TMainForm::JobTimerTimer(TObject *Sender)
{
     static const LPCOLESTR Competencies [] =
            {L"CORBA", L"COM+", L"MIDAS", L"ADO", L"VCL"};
     static const NCompetencies = sizeof(Competencies)/
            sizeof(Competencies [0]);
     WideString MainCompetence =
            Competencies [rand()%NCompetencies];
     OLECHECK(FJobEvent.JobAvailable(MainCompetence));
}
```

Теперь можно скомпилировать и сохранить файлы созданного приложения. Оно будет выполнять функции публикатора события — с интервалом в 5 секунд извещать подписчиков о появлении новой вакансии в агентстве по найму. Но если сейчас запустить эту программу на выполнение, ни-

каких видимых проявлений ее деятельности вы не заметите. Объяснение этому лежит на поверхности: никто не "оформил" подписку на сообщения нашего публикатора, а следовательно, некому реагировать на его предложения, какими бы привлекательными они не были.

Создание подписчиков

Естественно, что подписку могут "оформить" только подписчики, и пока мы не создадим хотя бы одного подписчика, о подписке не может быть и речи. В этом разделе будет показано, как создаются клиентские приложения подписчиков. Затем эти подписчики оформят подписку на сообщения созданного в предыдущем разделе публикатора, причем один из них оформит "постоянную" (persistent) подписку, а другой — "временную" (transient).

В приложениях подписчиков должен быть реализован интерфейс события. Первое приложение при получении сообщения `JobAvailable()` будет выводить на экран окно сообщения, поэтому в нем нужно реализовать интерфейс `IJobEvent`. В окне сообщения будет выводиться текст, переданный аргументом `MainCompetence` поступившего вызова.

Другое приложение подписчика будет использовать выходной интерфейс и точку подключения и переправлять сообщение о событии `JobAvailable()` клиенту, подключенному к приложению. Конечно, это приложение подписчика должно содержать реализацию интерфейса `IJobEvent`, но оно также будет содержать и реализацию выходного интерфейса, с помощью которого поступившее сообщение `JobAvailable()` перенаправляется клиентам. Таким образом, в этих приложениях мы продемонстрируем, как комбинировать в программе механизмы свободного и жесткого связывания событий.

На заметку Механизм жесткого связывания событий посредством точек подключения рассмотрен в главе 16.

Поскольку в обоих приложениях подписчиков используется один и тот же интерфейс класса `JobEvent`, воспользуемся имеющейся в составе C++Builder 5 функцией внедрения ранее созданного интерфейса.

Реализация интерфейса `IJobEvent`

Сначала с помощью C++Builder создайте новую библиотеку ActiveX и сохраните проект под именем `JobSubscribers`.

1. Выберите в главном меню C++Builder команду File⇨New. На экране откроется диалоговое окно New Items.
2. Выберите в этом окне вкладку ActiveX и дважды щелкните на ярлыке COM Object. Теперь на экране появится диалоговое окно New COM Object.
3. Установите а поле CoClass Name имя сопряженного класса **`PopupSubscriber`**, а в поле Threading Model — нейтральную модель потоков **`Neutral`**. Обратите внимание на то, что в поле Implemented Interface (Реализованный интерфейс) выведен текст `IPopupSubscriber`.
4. Поскольку нам нужен не новый интерфейс `IPopupSubscriber`, а уже существующий `IJobEvent`, щелкните на кнопке List. Обратите внимание: название диалогового окна после этого изменится на Scanning the Registry for Interfaces (Просмотр интерфейсов в реестре). Подождите немного и через несколько секунд на экране появится диалоговое окно мастера выбора интерфейсов Interface Selection Wizard. В этом диало-

говом окне перечислены все интерфейсы, определенные в библиотеках типов, зарегистрированных в системном реестре.

5. Отыщите в этом списке элемент интерфейса `jobevents.dll::IJobEvent` и щелкните на кнопке Select (рис. 18.11).
6. На сей раз в окне New COM Object в качестве интерфейса по умолчанию для COM-объекта `PopupSubscriber` будет предложен интерфейс `IJobEvent` (рис. 18.12). Щелкните на кнопке OK.

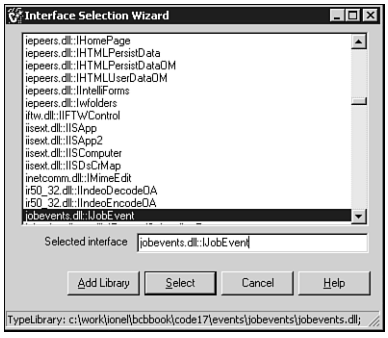


Рис. 18.11. Диалоговое окно Interface Selection Wizard, в котором выбран интерфейс `jobevents.dll::IJobEvent`

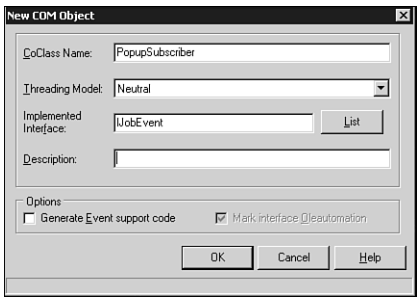


Рис. 18.12. Диалоговое окно New COM Object, в котором в качестве интерфейса по умолчанию для COM-объекта `PopupSubscriber` предлагается ранее сформированный интерфейс `IJobEvent`

7. Теперь на экране появится окно редактора библиотеки типов TLE. Сохраните файлы проекты. Обратите при этом внимание на то, что в проект добавлен новый файл `PopupSubscriberImpl.cpp`.
8. И еще одна деталь: в состав проекта следует включить файл `JobEvents_TLB.cpp`, который должен появиться в том же подкаталоге, что и остальные файлы проекта. Этот файл создан в результате неявного импортирования библиотеки типов `JobEvents` в тот момент, когда в состав COM-объекта включался интерфейс `IJobEvent`.

Итак, подведем итог. Мы располагаем заготовкой COM-объекта `PopupSubscriber`, который объявлен в файле `PopupSubscriberImpl.h`. Откройте этот файл и убедитесь, что программный код в нем уже вам известен. Этот программный код сформирован ATL, и вы не раз уже видели его в объявлении других COM-объектов.

Повторите первые пять операций описанной процедуры и сформируйте второй объект подписчика. Назовите его **`ForwardSubscriber`**. Но не забудьте об одном нюансе: при выполнении операции п. 4 установите флажок Generate Event Support Code (Генерировать код обработки события). Тем самым вы дадите знать C++Builder, что нужно сформировать программный код ATL, необходимый для возбуждения события выходного интерфейса диспетчеризации. Надеюсь, вы еще не забыли, что этот подписчик должен переадресовывать полученные сообщения о событии COM+ клиентам, связанным с ним через точки подключения.

На заметку

Интерфейсы диспетчеризации — не стандартные COM-интерфейсы, а командные, доступ к которым возможен из программ, написанных на языках создания сценариев. Интерфейсы диспетчеризации являются производными от базового интерфейса `IDispatch`.

После выполнения описанных операций проверьте, добавлен ли в состав проекта файл ForwardSubscriberImpl.cpp. В окне редактора библиотеки типов TLE вы должны увидеть то, что показано на рис. 18.13. Обратите внимание: сопряженный класс ForwardSubscriber реализует и интерфейс IJobEvent, и выходной интерфейс диспетчеризации IForwardSubscriberEvents.

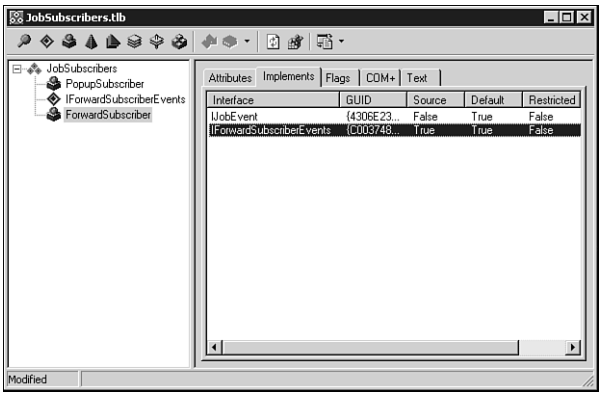


Рис. 18.13. Состояние окна редактора библиотеки типов TLE после добавления двух объектов подписчика

Интерфейс IForwardSubscriberEvents добавлен в сопряженный класс ForwardSubscriber, поскольку был установлен флажок Generate Event Support Code (Генерировать код обработки события). Добавьте в объявление интерфейса диспетчеризации следующий метод:

```
HRESULT JobOfferReceived([in] BSTR MainCompetence);
```

На заметку Методика добавления нового метода в объявление интерфейса диспетчеризации IForwardSubscriberEvents описана в главе 16.

Сохраните проект. Теперь у нас есть заготовка — скелет — приложений, но на эти "кости" еще предстоит нарастить "мясо".

Реализация метода IJobEvent::JobAvailable()

Оба приложения подписчиков должны реализовать один и тот же интерфейс IJobEvent, но реализовать по-разному.

Начнем с первого приложения — PopupSubscriber. Откройте файл PopupSubscriberImpl.cpp и включите в него следующий программный код реализации метода JobAvailable():

```
STDMETHODIMP TPopupSubscriberImpl::JobAvailable(
                                    BSTR MainCompetence)
{
   ShowMessage(MainCompetence);
   return S_OK;
}
```

При такой реализации метода на экран будет выводиться окно сообщения при получении каждого очередного извещения о событии.

Теперь займемся другим объектом подписчика — ForwardSubscriber. Этот объект будет переадресовывать полученное сообщение клиентам, связанным с ним через точки подключе-

ния. Включите в файл ForwardSubscriberImpl.cpp следующий программный код реализации метода JobAvailable():

```
STDMETHODIMP TForwardSubscriberImpl::JobAvailable(
                                        BSTR MainCompetence)
{
   Fire_JobOfferReceived(MainCompetence);
   return S_OK;
}
```

Метод Fire_JobOfferReceived() унаследован классом TForwardSubscriberImpl от класса TEvents_ForwardSubscriber<TForwardSubscriberImpl>. Этот программный код только оповещает о происшедшем событии COM+ клиентов, подключенных к методу IForwardSubscriberEvents::JobOfferReceived(). Точно такой же механизм оповещения клиентов использовался и в примерах, рассмотренных в главе 16. Если у вас возникли какие-то вопросы по поводу смысла происходящего в этой программе, перечитайте главу 16.

Сохраните файлы и скомпилируйте их. Далее мы займемся административными функциями.

Настройка конфигурации объектов подписчиков

Для настройки конфигурации обоих созданных объектов подписчиков выполните следующие операции.

1. Вызовите на экран окно Component Services с помощью меню панели задач операционной системы Windows 2000.
2. Разверните в левой панели окна ветвь Console Root и последовательно разворачивайте Component Services⇨My Computer⇨COM+ Applications.
3. Выделите узел COM+ Applications и создайте новое приложение **BCB Jobs Subscribers**.
4. Выделите в ветви приложения BCB Jobs Subscribers элемент Components и вызовите команду меню Action⇨New⇨Component. На экране появится первое окно мастера ***COM Component Install Wizard***. Щелкните на кнопке Next.
5. Теперь на экране появится следующее окно мастера (рис. 18.14. Щелкните на кнопке Install New Component(s).

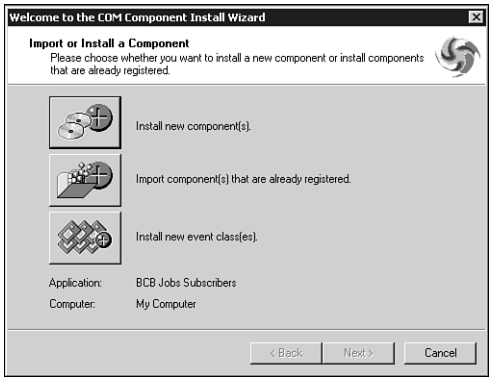


Рис. 18.14. Диалоговое окно Import or Install a Component *мастера* **COM Component Install Wizard**

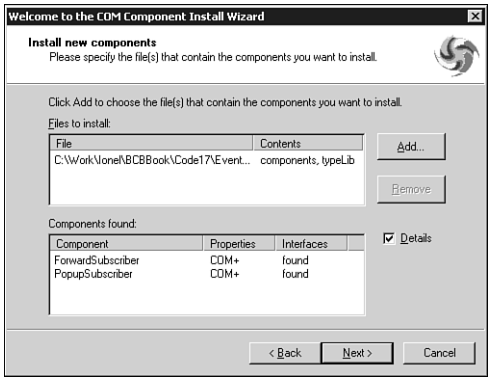


Рис. 18.15. Диалоговое окно Install New Components *мастера* **COM Component Install Wizard**

6. В открывшемся диалоговом окне Open File отыщите файл `JobSubscribers.dll`.
7. Дважды щелкните на имени файла. На экране появится новое диалоговое окно, в котором приведен список всех сопряженных классов, объявленных в COM-сервере `JobSubscribers.dll` (рис. 18.15).
8. Щелкните на кнопке Next, а в следующем окне — на кнопке Finish.

В результате всех этих манипуляций оба объекта подписчиков включены в новое приложение COM+.

Оформление „постоянной“ подписки

Нам требуется оформить “постоянную” подписку для подписчика `PopupSubscriber`, который желает получать уведомления о событии `JobEvent` от публикатора этого события. Выполните следующие операции.

1. В левой панели окна Component Services выберите элемент `Subscriptions` компонента `JobSubscribers.PopupSubscriber` приложения `BCB Jobs Subscribers` (рис. 18.16).

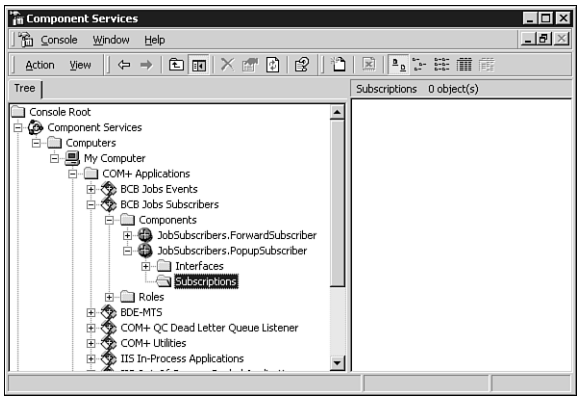


Рис. 18.16. Выбор элемента `Subscriptions` компонента `JobSubscribers.PopupSubscriber` приложения `BCB Jobs Subscribers` в диалоговом окне Component Services

2. Выберите в меню команду Action⇨New⇨Subscription. Теперь на экране откроется первое окно мастера ***COM New Subscription Wizard***. Щелкните на кнопке Next.
3. Во втором окне матера, Select Subscription Method(s), перечислены все интерфейсы, реализованные в объекте подписчика. В данном случае набор небогат — всего один интерфейс `IJobEvent` (рис. 18.17). Выделите этот элемент в списке, организовав таким образом подписку на все методы событий этого интерфейса. Щелкните на кнопке Next.
4. Теперь на экране появится следующее окно мастера, Select Event Class. В этом окне перечислены идентификаторы Prog ID всех зарегистрированных классов, реализующих интерфейс `IJobEvent`. В данном случае имеется только один такой класс `JobEvents.JobEvent` (рис. 18.18). Выделите его и щелкните на кнопке Next.

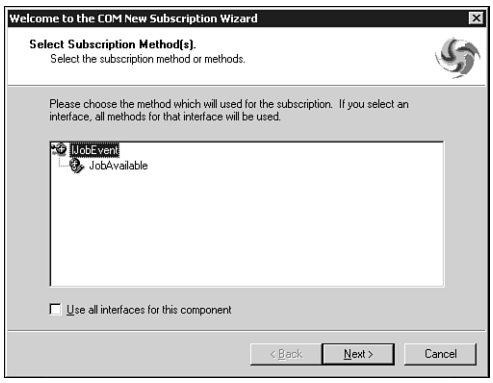


Рис. 18.17. Диалоговое окно Select Subscription Method(s) *мастера* **COM New Subscription Wizard**

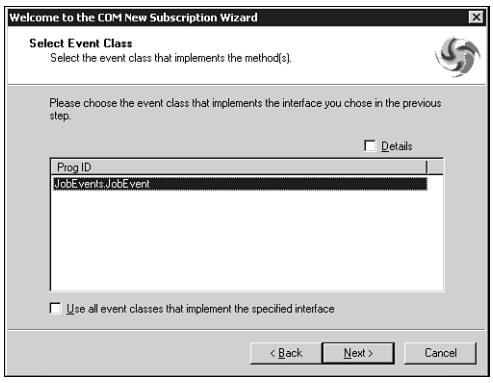


Рис. 18.18. Диалоговое окно Select Event Class *мастера* **COM New Subscription Wizard**

5. Следующее окно мастера — Subscriptions Options (рис. 18.19). В нем требуется задать имя подписки (назовем ее `Jobs`) и указать, будет ли приложение COM+ реагировать на эту подписку сразу же после завершения "оформления" (установите флажок Enable This Subscription Immediately). Щелкните на кнопке Next, а в следующем окне мастера — на кнопке Finish.

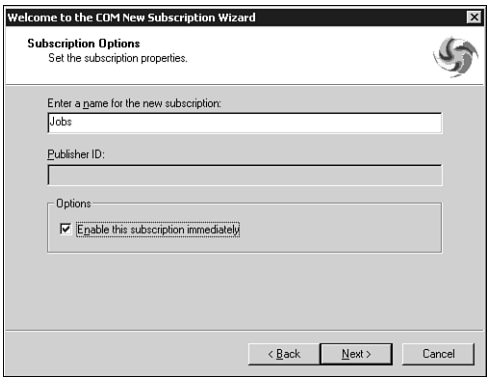


Рис. 18.19. Диалоговое окно Subscription Options *мастера* **COM New Subscription Wizard**

Вот теперь уже можно запустить на выполнение ранее созданную программу публикатора `JobPublisher.exe`. Понаблюдайте за тем, как каждые 5 сек на экране появляется окно сообщения, в котором выводится значение аргумента `MainCompentence` возбужденного события. Учтите, что подписчики и публикатор при этом даже не знают о существовании друг друга (т.е. нет никаких указаний на это в программном коде обоих приложений). Функции связи между публикатором и подписчиком выполняют средства поддержки COM+. Несмотря на то что она изобретена в Microsoft, модель COM+ работает прекрасно!

Теперь нам нужно организовать фильтрацию событий по значению.

1. В левой панели диалогового окна Component Services щелкните правой кнопкой мыши на только что сформированном элементе подписки `Jobs` и выберите в контек-

стном меню команду Properties (Свойства). На экране откроется диалоговое окно Jobs Properties.

2. Выберите к этом окне вкладку Options (Параметры) и введите в поле Filter Criteria следующее условие фильтрации: `MainCompetence == "COM+"` (рис. 18.20). Щелкните на кнопке OK и закройте диалоговое окно.

Если теперь запустить программу `JobPublisher.exe` (и набраться терпения), то через некоторое время появится сообщение о вакансии для специалиста по COM+. Другие предложения фильтр не пропустит, хотя сервер их и формирует.

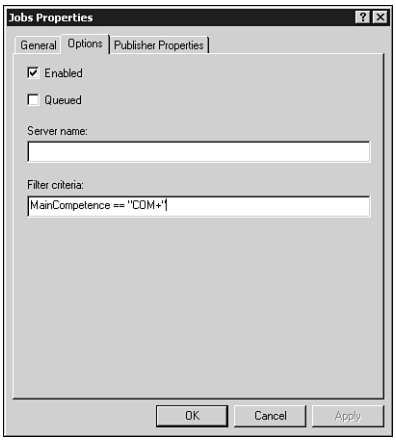


Рис. 18.20. Задание условия фильтрации по значению для подписки `Jobs`

Формирование „временной“ подписки

Временная (*transient*) подписка “оформляется” в программе без привлечения внешних средств администрирования. Для “оформления” временной подписки необходимо из программы обратиться к каталогу COM+ и работать с ним напрямую.

Ранее уже говорилось о том, что объект `ForwardSubscriber` будет использоваться для переадресации поступающих сообщений о событиях `IJobEvent::JobAvailable()` подключенным к этому объекту клиентам через выходной интерфейс. Поэтому в объекте `ForwardSubscriber` объявлен интерфейс диспетчеризации `IForwardSubscriberEvents`. На рис. 18.21 схематически показано, как механизм свободно связанных событий COM+ сочетается с механизмом жестко связанных событий, использующих точки подключения.

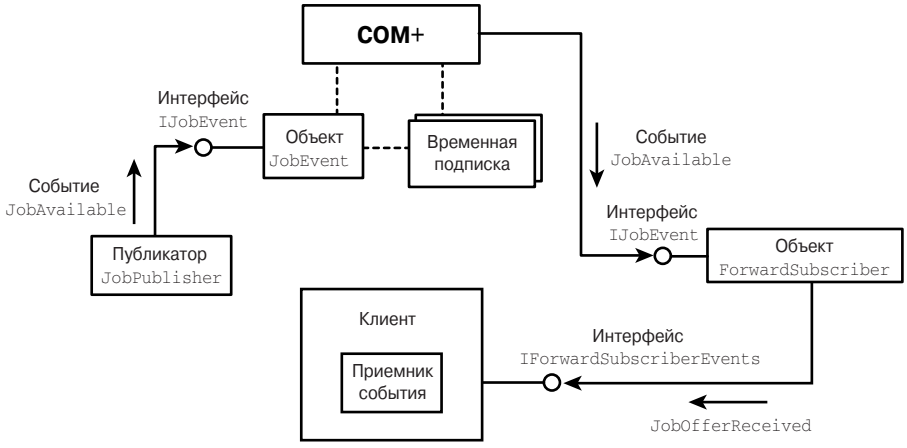


Рис. 18.21. Переадресация событий COM+ клиентам, связанным с подписчиком через точки подключения

Не будем занимать ваше время описанием процесса разработки программы, а рассмотрим ее основные фрагменты. Полный набор файлов приложения вы найдете в папке `TransientClient`

на прилагаемом компакт-диске. Там же находится и файл проекта TransientClient.bpr. Все приведенные ниже фрагменты взяты из файлов, включенных в состав этого проекта.

Формирование экземпляра класса подписчика

Будем считать, что создано приложение, главная экранная форма которого называется MainForm, а программный код реализации этого приложения находится в файле Main.cpp. Далее выполняются операции, которые уже знакомы вам по предыдущим разделам.

1. Импортируйте библиотеку типов сервера JobSubscribers. Для этого нужно выбрать в меню C++Builder команду Project⇨Import Type Library.

На заметку

Библиотека типов JobSubscribers зависит от библиотеки типов JobEvents. Поэтому C++Builder формирует два файла — JobSubscribers_TLB.cpp и JobEvents_TLB.cpp, но последний не включается автоматически в состав файлов проекта. Это вам придется сделать самостоятельно. При этом не держите зла на C++Builder — могу вас заверить, что, работая с Visual C++, вы столкнетесь еще не с такими казусами.

2. Включите в файл заголовка главной экранной формы директиву #include "JobSubscribers_TLB.h".
3. В определение класса TMainForm включите в качестве члена этого класса объект TCOMIJobEvent:

```
#include "JobSubscribers_TLB.h"

[ Пропущено... ]

class TMainForm :public TForm
{

   [ Пропущено... ]

   private:  // Объявления, вносимые разработчиком вручную
   TCOMIJobEvent FForwardSubscriber;
};
```

4. В программу обработки события FormCreate() включите оператор, формирующий экземпляр класса ForwardSubscriber:

```
FForwardSubscriber = CoForwardSubscriber::Create();
```

Подключение к ForwardSubscriber

Теперь в нашем распоряжении экземпляр сопряженного класса ForwardSubscriber. Создадим приемник события, который будет получать уведомления о событиях, возбужденных объектом ForwardSubscriber, через выходной интерфейс IForwardSubscriberEvents. В главе 16 уже рассматривалась методика создания приемника события, поэтому ограничимся только "цитированием" его программного кода в листинге 18.1. Сохраните этот код в файле FwSubSink.h.

Листинг 18.1. Реализация приемника события COM в интерфейсе IForwardSubscriberEvents

```
#if !defined(FWSUBSINK_H__)
#define FWSUBSINK_H__
#include <atlvcl.h>
```

```
#include <atlbase.h>
#include <atlcom.h>.
#include <ComObj.HPP>
#include <utilcls.h>
#include "JobSubscribers_TLB.h "
typedef void __fastcall (__closure *TForwardSubscriberEvent)
       (BSTR.MainCompetence);
//---------------------------------------------------------
// Класс, обслуживающий IForwardSubscriberEvents
class TForwardSubscriberSink :
    public TEventDispatcher<TForwardSubscriberSink,
           &DIID_IforwardSubscriber.Events>
{
   protected:
   // Поле события
   TForwardSubscriberEvent FOnJobAvailable;

   // Диспетчер события
   HRESULT InvokeEvent(DISPID id,TVariant*params)
   {
       if ((id ==1)&&(FOnJobAvailable != NULL))  //OnJobAvailable
           FOnJobAvailable(params [0 ]);
       return S_OK;
   }

   // Ссылка на источник события
   CComPtr<IUnknown> m_pSender;

   public:
       __property TForwardSubscriberEvent OnJobAvailable =
           {read = FOnJobAvailable, write = FOnJobAvailable };

   public:
      TForwardSubscriberSink():
            m_pSender(NULL),
            FOnJobAvailable(NULL)
      {
      }

      virtual ~TForwardSubscriberSink()
      {
         Disconnect();
      }

      // Подключение к серверу
      void Connect(IUnknown*pSender)
      {
         if (pSender !=m_pSender)
              m_pSender =pSender;
         if (NULL !=m_pSender)
```

```
            ConnectEvents(m_pSender);
        }

        // Отключение от сервера
        void Disconnect()
        {
            if (NULL !=m_pSender)
            {
                DisconnectEvents(m_pSender);
                m_pSender =NULL;
            }
        }
};
#endif//FWSUBSINK_H__
```

Теперь организуем связь с ForwardSubscriber с помощью точки подключения.

1. Вставьте в файл Main.h директиву включения файла FwSubSink.h.
2. Объявите в качестве членов класса экранной формы TMainForm объект класса TForwardSubscriberSink и обработчик события OnJobAvailable():

```
#include "JobSubscribers_TLB.h "
#include "FwSubSink.h "
      [ Пропущено... ]
class TMainForm :public TForm
{
      [ Пропущено... ]
    private://Объявления, вносимые разработчиком вручную
    TCOMIJobEvent FForwardSubscriber;
    TForwardSubscriberSink FForwardSubscriberSink;
    void __fastcall OnJobAvailable(BSTR MainCompetence);
};
```

3. Подключите приемник события к объекту ForwardSubscriber. Для этого в конец текста обработчика события FormCreate() добавьте следующий фрагмент:

```
FForwardSubscriberSink.OnJobAvailable = OnJobAvailable;
FForwardSubscriberSink.Connect(FForwardSubscriber);
```

4. Реализуйте по своему усмотрению обработчик события OnJobAvailable(). Это можно сделать, например, так, как показано в приведенном ниже фрагменте (в этом фрагменте MemoJobs — объект элемента управления memo):

```
void __fastcall TMainForm::OnJobAvailable(BSTR MainCompetence)
{
      // Зарегистрировать поступившее предложение
      MemoJobs->Lines->Add(MainCompetence);
}
```

Теперь клиентская программа готова к тому, чтобы оформить временную подписку с помощью объекта ForwardSubscriber.

Создание объекта временной подписки в каталоге COM+

Доступ к коллекциям объектов, хранящихся в каталоге COM+, организуется через интерфейс каталога. Для оформления новой временной подписки необходимо извлечь коллекцию объектов временной подписки из каталога, добавить в эту коллекцию новый объект, установить значения его свойств и сохранить внесенные изменения в каталоге COM+. Как все это делается, мы покажем ниже на примере из нашего приложения.

На заметку

В листингах программного кода. приведенных ниже, полужирным шрифтом будут выделены фрагменты, на которые нужно обратить особое внимание. Чтобы не повторяться, фрагменты, которые уже рассматривались ранее, в этих листингах будут опущены и заменены указателем `[ Пропущено... ]`.

Обращаю ваше внимание также на то, что приведенные фрагменты программного кода несколько упрощены, по сравнению с полным кодом приложения на прилагаемом компакт-диске.

1. Вставьте в файл `Main.cpp` директиву включения файла `<comadmin.h>`. В этом файле объявлены интерфейсы администрирования COM+, в том числе и интерфейс `ICOMAdminCatalog`.

На заметку

Если вы сейчас попробуете скомпилировать текст программы, то получите сообщение об ошибке:

```
comadmin.h(1899):E2146 Need an identifier to declare
```

```
comadmin.h(1899):E2146 Для объявления необходим идентификатор
```

Дважды щелкните на тексте сообщения и замените в строке программного кода завершающие символы `};` текстом **`} COMAdminErrorCodes;`**. Дело в том, что C++Builder не поддерживает неименованные директивы объявления типов `typedef`.

2. Измените текст обработчика события `FormCreate()`, как показано в следующем фрагменте. В этом фрагменте создается экземпляр объекта `COMAdmin.COMAdminCatalog` (корневого объекта в иерархии каталога COM+) и извлекается его интерфейс `ICOMAdminCatalog`:

```
void __fastcall TMainForm::FormCreate(TObject *Sender)
{
   // Создание объекта подписчика
   FForwardSubscriber = CoForwardSubscriber::Create();

   // Подключить приемник события к подписчику.
   FForwardSubscriberSink.OnJobAvailable = OnJobAvailable;
   FForwardSubscriberSink.Connect(FForwardSubscriber);

   // Извлечь каталог.
   CComPtr<ICOMAdminCatalog> ptrCatalog;
   OLECHECK(ptrCatalog.CoCreateInstance(
                  L"COMAdmin.COMAdminCatalog"));
}
```

3. Теперь можно извлечь коллекцию объектов временной подписки (имя коллекции — `TransientSubscriptions`). Добавьте в текст операторы, выделенные полужирным шрифтом:

```
void __fastcall TMainForm::FormCreate(TObject *Sender)
{
   [ Пропущено... ]

   // Извлечь каталог.
   CComPtr<ICOMAdminCatalog> ptrCatalog;
   OLECHECK(ptrCatalog.CoCreateInstance(
                  L"COMAdmin.COMAdminCatalog"));

   // Извлечь коллекцию TransientSubscriptions.
   CComPtr<IDispatch> ptrDisp;
   OLECHECK(ptrCatalog->GetCollection(
              WideString("TransientSubscriptions"),
              &ptrDisp));
   CComQIPtr<ICatalogCollection, &IID_ICatalogCollection> ptrCol =
                    ptrDisp.p;
   ATLASSERT(ptrCol != NULL);
}
```

Метод `GetCollection()` позволяет получить интерфейс `IDispatch` любого объекта коллекции каталога. Обратите внимание на использование объекта типа `WideString`. Этот объект используется при обращении к методу `GetCollection()`, поскольку последний получает имя извлекаемой коллекции через аргумент типа `BSTR`. Затем для запроса интерфейса `ICatalogCollection` используется объект класса `CComQIPtr`.

4. Получив ссылку на интерфейс `ICatalogCollection` объекта коллекции, можно обратиться к его методу `Add()` и создать новый объект временной подписки. Затем опять с помощью `CComQIPtr` запрашивается интерфейс, но на этот раз другой — `ICatalogObject` только что созданного объекта подписки. Все это выполняется в следующем фрагменте программы:

```
void __fastcall TMainForm::FormCreate(TObject *Sender)
{
      [ Пропущено... ]
   // Извлечь коллекцию TransientSubscriptions.
   CComPtr<IDispatch> ptrDisp;
   OLECHECK(ptrCatalog->GetCollection(
             WideString("TransientSubscriptions "),
             &ptrDisp));
   CComQIPtr<ICatalogCollection, &IID_ICatalogCollection> ptrCol =
             ptrDisp.p;
   ATLASSERT(ptrCol != NULL);

   // Добавить новый объект временной подписки.
   ptrDisp = NULL;
   OLECHECK(ptrCol->Add(&ptrDisp));
   ATLASSERT(ptrDisp != NULL);
   CComQIPtr<ICatalogObject, &IID_ICatalogObject> ptrObj =
             ptrDisp.p;
   ATLASSERT(ptrObj != NULL);
}
```

5. Используя интерфейс ICatalogObject, можно приступить к присвоению значений свойствам нового объекта подписки. Метод set_Value() позволяет указать имя свойства. Сначала установим интерфейс подписчика, который будет получать поступившие сообщения о событиях. В данном случае подписчиком является FForwardSubscriber:

```
void __fastcall TMainForm::FormCreate(TObject *Sender)
{
        [ Пропущено... ]

   // Добавить новый объект временной подписки.
   ptrDisp = NULL;
   OLECHECK(ptrCol->Add(&ptrDisp));
   ATLASSERT(ptrDisp != NULL);
   CComQIPtr<ICatalogObject,&IID_ICatalogObject> ptrObj =
            ptrDisp.p;
   ATLASSERT(ptrObj != NULL);

   // Установить интерфейс объекта подписчика, который будет
   // получать поступившие сообщения о событиях.
   OLECHECK(ptrObj->put_Value(WideString("SubscriberInterface"),
             CComVariant((IDispatch*)FForwardSubscriber)));
}
```

6. Далее установим идентификатор CLSID объекта того события, на которое нужно "оформить" подписку. Возможно, вас несколько удивит приведенный ниже программный код, но дело в том, что второй аргумент метода put_Value() объявлен как имеющий тип VARIANT, в потому необходимо преобразовать типы. Основную работу выполняют операторы, выделенные к этом фрагменте полужирным шрифтом.

```
void __fastcall TMainForm::FormCreate(TObject *Sender)
{
        [ Пропущено... ]

   // Установить интерфейс объекта подписчика, который будет
   // получать поступившие сообщения о событиях.
   OLECHECK(ptrObj->put_Value(WideString("SubscriberInterface"),
            CComVariant((IDispatch*)FForwardSubscriber)));

   // Установить CLSID объекта, к которому нужно подключиться.
   CComVariant vCLSID(GUIDToString(CLSID_JobEvent).c_str());
   vCLSID.ChangeType(VT_BSTR, &vCLSID);
   OLECHECK(ptrObj->put_Value(WideString("EventCLSID"), vCLSID));
}
```

7. Далее установим наименование объекта подписки. В качестве такового можно использовать любое имя, которое пришлось вам по душе:

```
void __fastcall TMainForm::FormCreate(TObject *Sender)
{
          [ Пропущено... ]

   // Установить CLSID объекта, к которому нужно подключиться.
   CComVariant vCLSID(GUIDToString(CLSID_JobEvent).c_str());
```

```
   vCLSID.ChangeType(VT_BSTR,&vCLSID);
   OLECHECK(ptrObj->put_Value(WideString("EventCLSID "),vCLSID));

   // Установить имя объекта подписки. Можно использовать любое.
   // Будет использован тот же идентификатор CLSID,
   // что и в предыдущей установке.
   OLECHECK(ptrObj->put_Value(WideString("Name"), vCLSID));
}
```

8. Свойству Enabled присвойте значение **VARIANT_TRUE**:

```
void __fastcall TMainForm::FormCreate(TObject *Sender)
{
       [ Пропущено... ]

   // Установить имя объекта подписки. Можно использовать любое.
   // Будет использован тот же идентификатор CLSID,
   // что и в предыдущей установке.
   OLECHECK(ptrObj->put_Value(WideString("Name"),vCLSID));

   // Разрешить подписку.
   OLECHECK(ptrObj->put_Value(WideString("Enabled"),
              CComVariant(VARIANT_TRUE)));
}
```

9. Когда в дальнейшем нужно будет "аннулировать" подписку и удалить ее объект из коллекции, нам понадобится идентификатор этого объекта. Объявите в классе TMainForm член FKey, в котором будет храниться значение идентификатора созданного объекта временной подписки:

```
class TMainForm :public TForm
{
   private:// Объявления, вносимые разработчиком вручную

       [ Пропущено... ]

   CComVariant FKey;
};

void __fastcall TMainForm::FormCreate(TObject *Sender)
{
      [ Пропущено... ]

   // Разрешить подписку.
   OLECHECK(ptrObj->put_Value(WideString("Enabled"),
        CComVariant(VARIANT_TRUE)));

   // Запомнить идентификатор созданного объекта.
   OLECHECK(ptrObj->get_Key(&FKey));
}
```

10. Теперь нужно сохранить внесенные изменения в каталоге COM+. Для этого необходимо вызвать метод ICatalogCollection::SaveChanges():

```
void __fastcall TMainForm::FormCreate(TObject *Sender)
{
      [ Пропущено... ]

   // Запомнить идентификатор созданного объекта.
   OLECHECK(ptrObj->get_Key(&FKey));

   // Сохранить изменения в каталоге.
   long id = 0;       // Не обращайте на это внимания.
   OLECHECK(ptrCol->SaveChanges(&id));
}
```

Удаление объекта временной подписки

Теперь займемся фрагментом программы, который организует "аннулирование" подписки (на то она и временная). Для "аннулирования" — удаления объекта подписки из каталога COM+ — нам понадобится идентификатор объекта, сохраненный ранее в члене-переменной `FKey` объекта экранной формы. Поскольку здесь не обойтись без "услуг" Microsoft API, то для организации удаления придется выполнить довольно много операций (такое всегда случается, если обращаешься к Microsoft API).

1. Добавьте в класс экранной формы приложения обработчик события `FormDestroy()`. Включите в него тот же программный код, который использовался ранее в обработчике `FormCreate()` для извлечения коллекции объектов подписки из каталога COM+.:

```
void __fastcall TMainForm::FormDestroy(TObject *Sender)
{
   // Извлечь каталог.
   CComPtr<ICOMAdminCatalog> ptrCatalog;
   OLECHECK(ptrCatalog.CoCreateInstance(
         L"COMAdmin.COMAdminCatalog "));

   // Извлечь коллекцию TransientSubscriptions.
   CComPtr<IDispatch> ptrDisp;
   WideString wstrCollName =
            L"TransientSubscriptions";
   OLECHECK(ptrCatalog->GetCollection(
                  wstrCollName, &ptrDisp));
   CComQIPtr<ICatalogCollection,&IID_ICatalogCollection> ptrCol =
             ptrDisp.p;
}
```

2. Созданный объект коллекции каталога не содержит никаких данных. Заполнение объекта выполняется методами `Populate()` или `PopulateByKey()`. В данном случае лучше использовать метод `PopulateByKey()`, поскольку желательно загрузить в кэш только тот объект, который планируется удалить, а ключ этого объекта известен. Метод `PopulateByKey()` получает в качестве аргумента вызова массив ключей тех объектов, которые нужно извлечь из каталога и включить в созданный объект коллекции. Поэтому в программе создается массив `SAFEARRAY`, в этот массив добавляется ключ, а затем вызывается метод `PopulateByKey()`, и объект коллекции заполняется данными:

```
void __fastcall TMainForm::FormDestroy(TObject *Sender)
{
```

```
        [ Пропущено... ]

    // Создание массива SAFEARRAY ранее запомненных
    // в программе ключей.
    SAFEARRAY* pKeys = SafeArrayCreateVector(VT_VARIANT, 0, 1);
    long ix[1];
    ix[0] = 0;
    SafeArrayPutElement(pKeys, ix, &FKey);
    OLECHECK(ptrCol->PopulateByKey(pKeys));
    SafeArrayDestroy(pKeys);
}
```

3. Теперь удалим объект подписки, который соответствует ранее запомненному идентификатору:

```
void __fastcall TMainForm::FormDestroy(TObject *Sender)
{
        [ Пропущено... ]

    // Создание массива SAFEARRAY ранее запомненных
    // в программе ключей.
    SAFEARRAY* pKeys = SafeArrayCreateVector(VT_VARIANT, 0, 1);
    long ix[1];
    ix[0] = 0;
    SafeArrayPutElement(pKeys, ix, &FKey);
    OLECHECK(ptrCol->PopulateByKey(pKeys));
    SafeArrayDestroy(pKeys);

    // Удалить объект временной подписки, который является
    // единственным объектом в этой коллекции.
    long Count = 0;
    OLECHECK(ptrCol->get_Count(&Count));
    ATLASSERT(Count == 1);
    OLECHECK(ptrCol->Remove(0));
}
```

4. Осталось только зафиксировать внесенные изменения в каталоге COM+.

```
void __fastcall TMainForm::FormDestroy(TObject *Sender)
{
      [ Пропущено... ]

    // Удалить объект временной подписки, который является
    // единственным объектом в этой коллекции.
    long Count = 0;
    OLECHECK(ptrCol->get_Count(&Count));
    ATLASSERT(Count == 1);
    OLECHECK(ptrCol->Remove(0));

    // Зафиксировать изменения в каталоге COM+.
    long id = 0; // Не обращайте на это внимания.
    OLECHECK(ptrCol->SaveChanges(&id));
}
```

Даже самая длинная дорога когда-то кончается. Вот и мы завершили работу над примером, демонстрирующим создание объекта временной подписки. Запустите программу клиента (ее выполняемый файл вы найдете на прилагаемом компакт-диске) и ранее созданный выполняемый файл провайдера. Проследите, как они будут реагировать на сообщения об открывшихся вакансиях. Но не очень засиживайтесь — нам еще предстоят другие неизведанные дороги.

Разработка и использование объектов транзакций COM+

Вы наверняка знакомы со множеством примеров использования транзакций в системах баз данных, но на сей раз мы рассмотрим эту тему с несколько иной точки зрения. В качестве примера используем довольно простое приложение, перемещающее данные между двумя составными (compound) файлами. Это приложение не только продемонстрирует, что транзакции можно применять в различных сценариях, но позволит вам увидеть, что происходит за кулисами этого представления. Приложение, которое будет рассматриваться в этом разделе, носит название `Compensated Resource Manager` (менеджер компенсированных ресурсов).

На заметку

Составной (compound) файл представляет собой *объект структурированного хранения данных* (structured storage object), который можно рассматривать как реализацию файловой системы внутри отдельного файла. В терминах хранения структурированных данных эквивалентом каталога является *хранилище* (storage), причем корневой объект — корневое хранилище — может содержать *подхранилища* (substorages). Аналогами файлов в системе хранения структурированных данных являются *потоки* (streams), которые и содержат данные. Потоки в древовидной иерархической структуре составного файла являются терминальными узлами.

Интерфейс `IStorage` предоставляет в распоряжение разработчика методы формирования и манипулирования объектами хранилищ (в том числе и вложенными) и потоков. В частности, они позволяют создавать, открывать, перечислять, перемещать, копировать, переименовывать и удалять элементы в любом объекте хранилища. Составной файл можно открыть в режиме выполнения транзакций, а интерфейс `IStorage` содержит методы `Commit()` и `Revert()`, с помощью которых можно зафиксировать внесенные транзакцией изменения в объекте хранилища или оставить этот объект в неприкосновенности. Эти методы будут довольно часто использоваться в рассматриваемом ниже приложении.

Если вы пожелаете более подробно познакомиться с составными файлами и методами работы с ними, обратитесь к любой из многочисленных книг, посвященных модели COM, и отыщите в ней раздел о хранении структурированных данных (Structured Storage). Информацию о составных файлах можно найти и в системе электронной справки MSDN. Я настоятельно рекомендую хотя бы в общих чертах познакомиться с этой темой, прежде чем продолжить чтение. Но должен сказать, что дальнейший материал этой главы вы сможете "осилить", и не вдаваясь глубоко в технологию работы с составными файлами. Мы будем использовать только отдельные методы интерфейса `IStorage`.

Для выполнения операции взаимного обмена данными между двумя составными файлами нам потребуется три совместно работающих объекта:

- `Eraser` (Чистильщик) — этот объект удаляет содержимое из составного файла;
- `Copier` (Копировщик) — копирует содержимое одного составного файла в другой;

- `Swapper` (Обменник) — использует два других объекта для взаимного обмена содержимым двух файлов. Объект формирует временный составной файл и копирует в него содержимое первого файла, затем с помощью `Eraser` очищает первый файл и копирует в него содержимое второго файла. Последний этап — удаление с помощью `Eraser` содержимого второго файла и копирование в него содержимого временного файла. Объект временного файла после этого уничтожается.

Совершенно очевидно, что если на одном из этапов этой процедуры возникнет какая-либо проблема и операция не будет завершена, содержимое одного из файлов будет утеряно. По этой причине нужно рассматривать комплексную процедуру обмена как единую операцию, которую нельзя выполнить частично. Либо эта операция должна быть доведена до конца, либо все должно остаться в исходном состоянии. Именно поэтому для выполнения этой операции идеально подходит механизм транзакций.

Формирование объектов транзакций

Объекты `Eraser`, `Copier` и `Swapper` представляют средний, промежуточный уровень в трехуровневой модели обработки информации. Этот уровень принято называть уровнем *бизнес-логики* (business logic). Нижний уровень этой модели получил наименование *уровня данных* (data layer) — его образуют объекты составных файлов, а верхний — *уровень представления* (presentation layer) — это клиентское приложение, которое обращается за "услугами" к уровню бизнес-логики.

Объекты `Swapper`, `Copier` и `Eraser` не имеют внутреннего состояния, т.е. при каждом очередном обращении выполняют операцию независимо от того, какая операция и как именно была выполнена перед этим. Обратите внимание на то, что даже имена объектов выбраны таким образом. что отображают только характер операции и никак не связаны с содержимым данных, которыми оперирует объект.

Уровень бизнес-логики создается следующим образом.

1. Запустите C++Builder и создайте новую библиотеку ActiveX (на вкладке ActiveX диалогового окна New Items дважды щелкните на ярлыке ActiveX Library). Сохраните проект под именем `StgBusiness`.
2. Выберите в системном меню File⇨New — на экране откроется диалоговое окно New Items.
3. Выберите в этом окне вкладку ActiveX и дважды щелкните на ярлыке Transactional Object. Теперь на экране откроется диалоговое окно New Transactional Object (рис. 18.22).
4. В поле CoClass Name задайте имя нового объекта **`StgSwapper`**. В зоне Transaction Model (Модель транзакции) установите переключатель Requires a Transaction (Требует транзакции) и щелкните на кнопке OK.

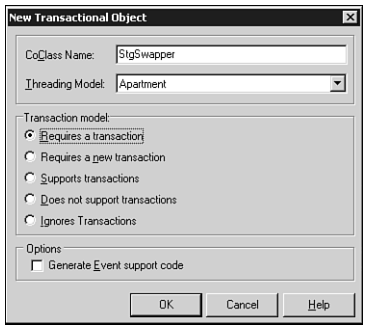


Рис. 18.22. Диалоговое окно New Transactional Object, с помощью которого создается объект транзакции

5. Повторите операции по пп. 1–4 и создайте таким же способом еще два объекта — `StgCopier` и `StgEraser`.

На заметку

Планируется, что в процессе работы объект StgSwapper будет создавать экземпляры объектов StgCopier и StgEraser. При создании всех трех объектов устанавливается переключатель Requires a Transaction. Если клиент создает объект StgSwapper вне контекста транзакции, COM+ самостоятельно формирует новую транзакцию и назначает ее контексту объекта StgSwapper. Когда объект StgSwapper создает экземпляры StgCopier и StgEraser, назначенная транзакция распространяется и на эти экземпляры. Таким образом, все три объекта работают в контексте одной и той же транзакции, назначенной объекту StgSwapper.

Сохраните проект. Обратите внимание на то, что для каждого созданного объекта транзакций в проект включается свой файл реализации.

C++Builder автоматически включает в состав каждого вновь созданного объекта транзакций интерфейс IObjectControl. Этот интерфейс необходим для работы с Microsoft Transaction Server (MTS) и крайне желателен для работы со средствами поддержки COM+. Пользуясь интерфейсом IObjectControl, COM+ оповещает об активизации и прекращении работы, при этом получатели сообщения могут организовать выделение и высвобождение ресурсов, если в этом есть необходимость. Интерфейс IObjectControl имеет три метода:

- HRESULT Activate() — метод вызывается средствами поддержки COM+ и переводит объект в активное состояние. Этот метод, как правило, инициализирует активизируемый объект.
- void Deactivate() — метод, обратный Activate(). Вызывается средствами поддержки COM+ и переводит объект в неактивное состояние. Обычно в этом методе программируются операции высвобождения занятых ресурсов.
- BOOL CanBePooled() — метод возвращает значение **TRUE**, если объект позволяет использовать пул объектов.

Теперь необходимо включить программный код в заготовки объектов транзакций, созданные C++Builder. Поскольку объекты не имеют внутреннего состояния, их можно помещать в пул объектов. Откройте файл реализации TStgSwapperImpl (файл типа .cpp) и измените программный код метода CanBePooled() — этот метод должен возвращать значение **TRUE**:

```
STDMETHODIMP_(BOOL)TStgSwapperImpl::CanBePooled()
{
    return TRUE;
}
```

Аналогичные изменения внесите и в файлы реализации классов TStgCopierImpl и TStgEraserImpl.

Далее с помощью редактора библиотеки типов TLE организуйте регистрацию всех трех классов в каталоге COM+ с разрешенным атрибутом Object Pooling. В качестве примера, покажем как это делается в отношении сопряженного класса StgSwapper. Выберите имя этого класса в левой панели редактора библиотеки типов TLE, а в правой — откройте вкладку COM+. Установите в ней флажок Object Pooling (рис. 18.23). Точно так же устанавливается атрибут и в классах StgEraser и StgSwapper.

Учтите, что в списке Call Synchronization на вкладке COM+ нужно выбрать режим Required — тогда в каталоге сопряженный класс будет зарегистрирован как требующий синхронизации со стороны COM+.

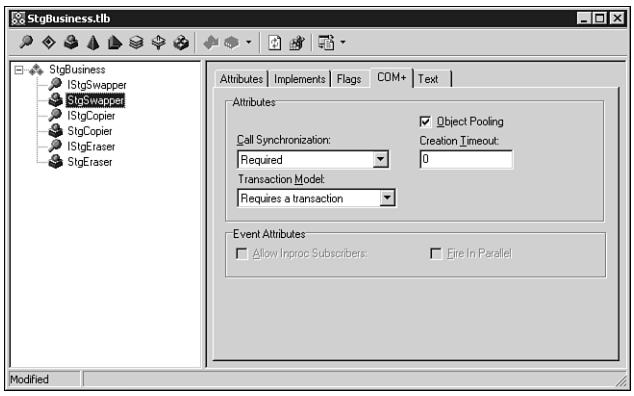


Рис. 18.23. Установка атрибутов сопряженного класса `StgSwapper` *на вкладке* COM+

Извлечение интерфейса `IObjectContext`

Интерфейс `IObjectContext` обеспечивает доступ к текущему контексту объекта. Хотя интерфейс располагает набором методов, чаще всего он используется для "участия в голосовании" по поводу успешного выполнения транзакции. Вызывая метод `IObjectContext::SetAbort()`, объект извещает компонент COM+, который координирует выполнение транзакций, — ***Distributed Transaction Coordinator*** (DTC) — о том, что транзакцию, в которую вовлечен данных объект, следует прервать. Другой метод, `IObjectContext::SetComplete()`, вызывается объектом в том случае, когда, "по мнению" данного объекта, транзакцию можно считать состоявшейся и ее результат следует зафиксировать.

В файле `StgSwapperImpl.cpp` метод `Activate()` класса `TStgSwapperImpl` реализован C++Builder следующим образом:

```
STDMETHODIMP TStgSwapperImpl::Activate()
{
   static TMtsDll Mts;
   HRESULT hr = E_FAIL;
   hr =Mts.Get_ObjectContext(&m_spObjectContext);
   if (SUCCEEDED(hr))
       return S_OK;
   return hr;
}
```

В этой программе объект получает интерфейс `IObjectContext` по отношению к экземпляру `StgSwapper`. Аналогичный фрагмент имеется и в файлах реализации `StgCopierImpl.cpp` и `StgEraserImpl.cpp`. Но этот код не работает, как задумано, под управлением COM+ (по крайней мере в операционной системе Windows 2000 Professional); вызов метода `Mts.Get_ObjectContext(&m_spObjectContext)` всегда заканчивается неудачно. Думаю, что в последующих версиях C++Builder эта ошибка будет исправлена. Вам следует заменить представленный ниже фрагмент метода `Activate()`

```
static TMtsDll Mts;
HRESULT hr =E_FAIL;
hr =Mts.Get_ObjectContext(&m_spObjectContext);
```

В исправленной версии нужно вызывать функцию CoGetObjectContext():

```
HRESULT hr =CoGetObjectContext(IID_IObjectContext,
reinterpret_cast<void**> (&m_spObjectContext));
```

При желании можно даже использовать директивы условной компиляции, поскольку функция CoGetObjectContext() доступна только в операционной системе Windows 2000.

Объявление специфических методов интерфейса по умолчанию для каждого объекта

В этом разделе будут определены методы, которые реализуют специфические функции объектов каждого из трех созданных типов. С помощью редактора библиотеки типов TLE включите в интерфейс IStgEraser следующий метод:

```
HRESULT _stdcall Erase([in] IUnknown **pStorage);
```

Этот метод будет очищать содержимое указанного объекта хранилища. Через аргумент pStorage передается указатель на интерфейс IStorage объекта хранилища, содержимое которого подлежит удалению. Если передать указатель на интерфейс IStorage корневого объекта хранилища составного файла, будет удалено содержимое всего файла.

На заметку Поскольку IStgEraser является дуальным интерфейсом, который допускает только аргументы, совместимые с механизмом автоматизации, при определении типа аргумента pStorage использован интерфейс Iunknown, вместо IStorage.

Теперь в интерфейс IStgCopier добавьте метод Copy():

```
HRESULT _stdcall Copy([in] IUnknown **pStgFrom,
                      [in] IUnknown **pStgTo);
```

Метод Copy() будет копировать содержимое одного объекта хранилища в другой. Первый аргумент метода является указателем на интерфейс IStorage объекта-источника, а второй — указателем на интерфейс IStorage объекта-приемника.

И наконец, в состав интерфейса IStgSwapper включим метод Swap():

```
HRESULT _stdcall Swap([in] BSTR File1, [in] BSTR File2);
```

Аргументами метода являются имена файлов, обменивающихся информацией.

Реализация объекта StgEraser

Начнем с реализации метода IStgEraser::Erase(). Откройте файл StgEraserImpl.cpp и вставьте в него директивы включения файла ComThrow.h (этот файл вы найдете на прилагаемом компакт-диске) и STL-файла <vector>. Текст метода TStgEraserImpl::Erase()представлен в листинге 18.2.

Листинг 18.2. Реализация метода TStgEraserImpl::Erase()

```
 [ Пропущено... ]

#include "ComThrow.h"
#include <vector>

[ Пропущено... ]
```

```
STDMETHODIMP TStgEraserImpl::Erase(LPUNKNOWN pStorage)
{
   HRESULT hResult = S_OK;
   try
   {
      ATLASSERT(m_spObjectContext != NULL);
      // Запрос интерфейса IStorage.
      CComQIPtr<IStorage> ptrStg = pStorage;
      BCHECK(ptrStg != NULL,E_NOINTERFACE);

      // Удаление содержимого объекта хранилища.
      STATSTG stat;
      CComPtr<IEnumSTATSTG> ptrEnum;
      HCHECK(ptrStg->EnumElements(0,NULL,0,&ptrEnum));
      HCHECK(ptrEnum->Reset());
      std::vector<LPOLESTR> vNames;
      while (ptrEnum->Next(1,&stat,NULL)==S_OK)
         vNames.push_back(stat.pwcsName);
      CComPtr<IMalloc> ptrMalloc;
      HCHECK(CoGetMalloc(1,&ptrMalloc));
      ATLASSERT(ptrMalloc != NULL);
      for (std::vector<LPOLESTR>::iterator it = vNames.begin();
         it != vNames.end(); ++it)
      {
         HCHECK(ptrStg->DestroyElement(*(it)));
         ptrMalloc->Free(*(it));   //CoTaskMemFree(*(it));
      }
      // Операция выполнена. "Голосуем" за фиксацию транзакции.
      m_spObjectContext->SetComplete();
   }

   catch(ECOMError &e)
   {
      // Возникла исключительная ситуация. Прервать операцию.
      m_spObjectContext->SetAbort();
      hResult =Error(e.Message,IID_IStgEraser,e.ErrorCode);
   }
   catch(...)
   {
      // Возникла исключительная ситуация. Прервать операцию.
      m_spObjectContext->SetAbort();
      hResult =E_UNEXPECTED;
   }
   return hResult;
};
```

Не будем вдаваться в детали операции удаления содержимого объекта хранилища, поскольку описание манипуляций с объектами этого типа выходит за рамки этой книги. Обратим внимание на методику использования структурированных исключений для обработки ошибок. Если все выполнено, как задумано, вызывается метод `m_spObjectContext->SetComplete()`.

На заметку Не забывайте, что объект m_spObjectContext настроен на интерфейс IObjectContext при выполнении метода Activate().

Если же в процессе выполнения возникает какая-либо ошибка, генерируется исключительная ситуация, для чего используются макросы BCHECK и HCHECK (эти макросы объявлены в файле ComThrow.h). В обоих блоках перехвата исключений catch вызывается метод m_spObjectContext->SetAbort(), который передает DTC сообщение, что текущая транзакция прервана.

Реализация объекта StgCopier

В методе IStgCopier::Copy() используется та же технология обработки исключительных ситуаций, что и рассмотренная выше. Практически мне нечего добавить к сказанному, поэтому самостоятельно просмотрите листинг 18.3, в котором представлен фрагмент файла StgCopierImpl.cpp с текстом программы этого метода.

Листинг 18.3. Реализация метода TStgCopierImpl::Copy()

```
 [ Пропущено... ]

#include "ComThrow.H".

STDMETHODIMP TStgCopierImpl::Copy(LPUNKNOWN pStgFrom,
            LPUNKNOWN pStgTo)
{
   HRESULT hResult = S_OK;
   try
   {
      ATLASSERT(m_spObjectContext != NULL);
      CComQIPtr<IStorage> ptrSrc = pStgFrom;
      BCHECK(ptrSrc != NULL,E_NOINTERFACE);
      CComQIPtr<IStorage> ptrDst = pStgTo;
      BCHECK(ptrDst != NULL,E_NOINTERFACE);

      // Копирование
      HCHECK(ptrSrc->CopyTo(NULL, NULL, NULL, ptrDst));
      #if defined(SIMULATE_ERROR)
         BCHECK((rand()%2)= 0, E_FAIL);
      #endif
      // Операция выполнена. "Голосуем" за фиксацию транзакции.
      m_spObjectContext->SetComplete();
   }

   catch(ECOMError &e)
   {
      // Возникла исключительная ситуация. Прервать операцию.
      m_spObjectContext->SetAbort();
      hResult =Error(e.Message, IID_IStgCopier, e.ErrorCode);
   }

   catch(...)
   {
```

```
      // Возникла исключительная ситуация. Прервать операцию.
      m_spObjectContext->SetAbort();
      hResult =E_UNEXPECTED;
   }
   return hResult;
};
```

В эту программу добавлен флаг условной компиляции SIMULATE_ERROR, с помощью которого в процессе отладки можно смоделировать случайное появление ошибки. Пользуясь им, можно просмотреть, как будет вести себя программа обработки транзакций в этой ситуации.

Реализация объекта StgSwapper

При выполнении своих функций объект StgSwapper использует объекты двух других типов. Текст программы, реализующей метод TStgSwapperImpl::Swap(), вы найдете в файле StgSwapperImpl.cpp (листинг 18.4). Обратите внимание на директиву включения файла ComThrow.h в самом начале StgSwapperImpl.cpp.

Листинг 18.4. Реализация метода TStgSwapperImpl::Swap()

```
STDMETHODIMP TStgSwapperImpl::Swap(BSTR File1, BSTR File2)
{
   HRESULT hResult = S_OK;
   try
   {
      ATLASSERT(m_ptrCrmWorker != NULL);
      ATLASSERT(m_spObjectContext != NULL);

      // Открыть оба файла.
      CComPtr<IStorage> ptrSrc;
      HCHECK(StgOpenStorageEx(File1,
           STGM_READWRITE|STGM_TRANSACTED|STGM_SHARE_DENY_WRITE,
            STGFMT_ANY,
            0,NULL, NULL,
            IID_IStorage,
            reinterpret_cast<void**> (&ptrSrc) ));

      CComPtr<IStorage> ptrDst;

      HCHECK(StgOpenStorageEx(File2,
         STGM_READWRITE | STGM_TRANSACTED | STGM_SHARE_DENY_WRITE,
         STGFMT_ANY,
         0, NULL, NULL,
         IID_IStorage,
         reinterpret_cast<void**> (&ptrDst) ));

      // Сформировать временный файл.
      CComPtr<IStorage> ptrTmp;

      HCHECK(::StgCreateDocfile(NULL,
         STGM_DELETEONRELEASE |STGM_CREATE |STGM_READWRITE |
```

```
            STGM_TRANSACTED |STGM_SHARE_EXCLUSIVE,
            0, &ptrTmp));

         // Создать объекты Copier и Eraser.
         CComPtr<IStgCopier> Copier;
         HCHECK(Copier.CoCreateInstance(CLSID_StgCopier));
         CComPtr<IStgEraser> Eraser;
         HCHECK(Eraser.CoCreateInstance(CLSID_StgEraser));

         // Выполнить обмен информацией.
         HCHECK(Copier->Copy(ptrDst,ptrTmp));
         HCHECK(Eraser->Erase(ptrDst));
         HCHECK(Copier->Copy(ptrSrc,ptrDst));
         HCHECK(Eraser->Erase(ptrSrc));
         HCHECK(Copier->Copy(ptrTmp,ptrSrc));

         // Операция выполнена. "Голосуем" за фиксацию транзакции.
         m_spObjectContext->SetComplete();
      }

      catch(ECOMError &e)
      {
         // Возникла исключительная ситуация. Прервать операцию.
         m_spObjectContext->SetAbort();
         hResult = Error(e.Message, IID_IStgSwapper, e.ErrorCode);
      }
      catch(...)
      {
         // Возникла исключительная ситуация. Прервать операцию.
         m_spObjectContext->SetAbort();
         hResult = E_UNEXPECTED;
      }
      return hResult;
}
```

Обратите внимание на то, как при вызове `StgOpenStorageEx()` используется флаг Notice that we used the `STGM_TRANSACTED`. Включение этого флага в аргумент вызова гарантирует, что составной файл будет открыт в режиме выполнения транзакций. Ниже будет показано, как использовать это свойство составного файла при его интегрировании в механизм выполнения транзакций компонента COM+ DTC.

Для инициирования объектов `Copier` и `Eraser` используется метод `CoCreateInstance()` класса `CComPtr`:

```
// Создать объекты Copier и Eraser.
CComPtr<IStgCopier> Copier;
HCHECK(Copier.CoCreateInstance(CLSID_StgCopier));
CComPtr<IStgEraser> Eraser;
HCHECK(Eraser.CoCreateInstance(CLSID_StgEraser));
```

Этот метод представляет собой альтернативное решение, прелагаемое ATL, вместо smart-интерфейсов C++Builder. Если применять "родные" smart-интерфейсы C++Builder, то соответствующий фрагмент программы будет выглядеть так:

```
// Создать объекты Copier и Eraser.
TCOMIStgCopier Copier = CoStgCopier::Create();
TCOMIStgEraser Eraser = CoStgEraser::Create();
```

Какой из вариантов выбрать — дело вкуса и личных предпочтений разработчика.

В остальной части метода используется та же технология работы с исключениями. Если в процессе выполнения операции возникает исключительная ситуация, то в блоке перехвата исключения транзакция прерывается. В противном случае — до исключительной ситуации дело не дошло — метод "голосует" за фиксацию транзакции.

После включения в состав программы описанных объектов и их методов можно считать разработку сервера `StgBusiness` завершенной. Теперь программа сервера готова к тому, чтобы экземпляр сервера можно было сформировать в составе приложения COM+.

Установка экземпляра сервера `StgBusiness` в новое приложение COM+

Создайте новое приложение COM+ и назовите его **`BCB Storage Objects`**. Установка экземпляра сервера `StgBusiness` в этом приложении выполняется по той же методике, что и установка сервера `JobSubscribers` в приложении `BCB Jobs Subscribers`. После того как процедура будет завершена, вы увидите в диалоговом окне Component Services то, что показано на рис. 18.24.

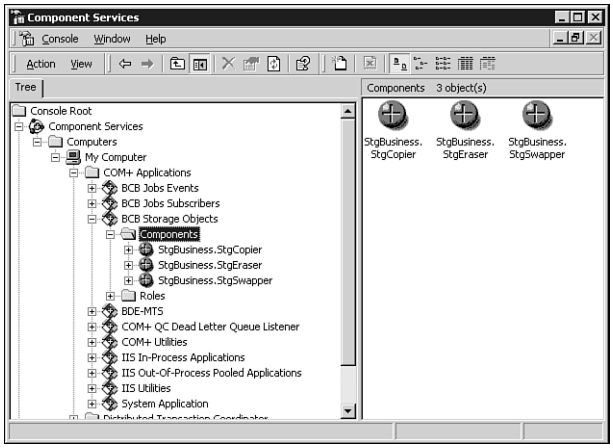


Рис. 18.24. Отображение в диалоговом окне Component Services объектов сервера `StgBusiness`, включенных в приложение `BCB Storage Objects`

Остался невыясненным существенный вопрос: какой компонент будет фиксировать или аннулировать транзакцию в соответствии с итогами "голосования" различных объектов, принимающих в ней участие?

Внимательно просмотрев тексты программ в cpp-файлах, вы не найдете в них вызовов методов `Commit()` или `Revert()` интерфейса `IStorage`. Именно эти методы должны выполнить на физическом уровне операции, связанные с фиксацией результатов транзакции в составных файлах или аннулированием транзакции и сохранением файлов в "первобытном состоянии". Реализуются эти методы еще одним программным компонентом COM+ — менеджером ресурсов ***Compensated Resource Manager*** (CRM). К серверу `StgBusiness` мы еще вернемся после создания собственного компонента CRM.

Разработка менеджеров ресурсов CRM

Разработка CRM — самый простой способ интеграции менеджера ресурсов и ***Distributed Transaction Coordinator*** (DTC). Собственно CRM состоит из двух субкомпонентов COM+ — ***CRM Worker*** и ***CRM Compensator***.

Субкомпонент ***CRM Worker*** должен поддерживать специализированные интерфейсы приложения и обеспечивать прием от клиентов команд, связанных с выполнением транзакций. Эти операции соответствуют виду изменений, вносимых в сохраняемые данные.

Клиент транзакции формирует экземпляр объекта ***CRM Worker*** и пользуется его инфраструктурой. После активизации объект ***CRM Worker*** формирует объект ***CRM Clerk***. Затем объект ***CRM Worker*** уведомляет с помощью метода `RegisterCompensator()` интерфейса `ICrmLogControl` созданного объекта ***CRM Clerk*** о соответствующем ему классе ***CRM Compensator***.

На заметку Объект ***CRM Clerk*** отвечает за выполнение внутренних операций формирования экземпляра данного класса ***CRM Compensator***.

После этого объект ***CRM Worker*** готов к тому, что им будет пользоваться клиентское приложение. Как только поступает команда от клиента, объект ***CRM Worker*** вносит в системный журнал регистрации запись об этой команде. Операция выполняется с помощью интерфейса `ICrmLogControl` "подчиненного" ему объекта ***CRM Clerk***.

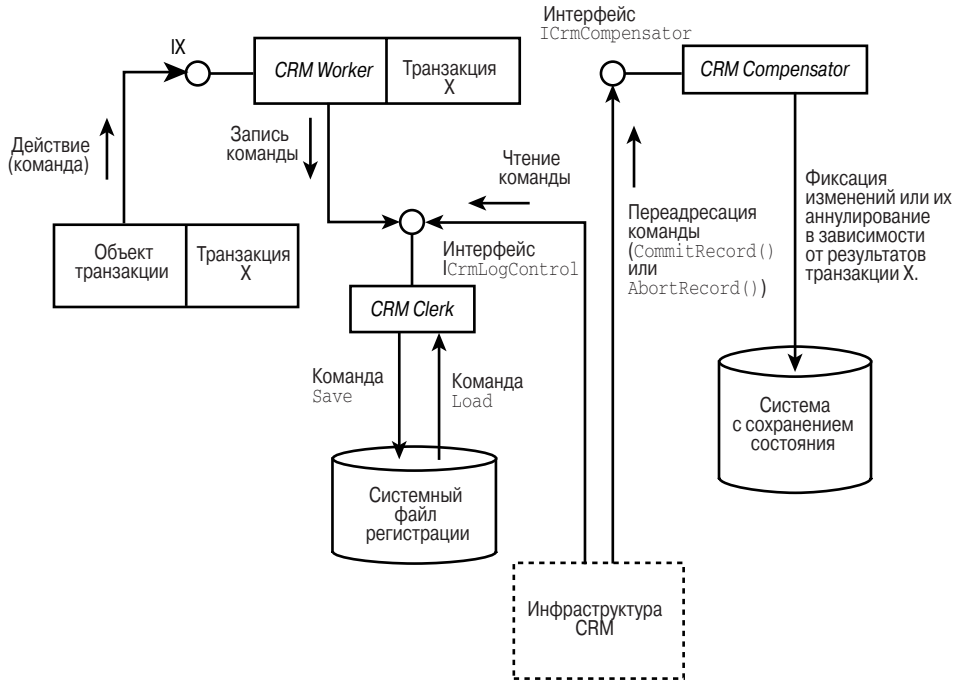


Рис. 18.25. Схема взаимодействия объектов **CRM Worker**, **CRM Clerk** *и* **CRM Compensator**

После этого на сцену выходит "партнер" ***CRM Worker*** — субкомпонент ***CRM Compensator***. Любой объект ***CRM Compensator***, разработанный на языке C++, реализует интерфейс `ICrmCompensator`. В этом интерфейсе имеются два метода, которые нас сейчас более всего ин-

тересуют, —CommitRecord() и AbortRecord(). Именно с помощью этих методов объект ***CRM Compensator*** уведомляет о фиксации или аннулировании транзакции в контексте, который сформирован объектом ***CRM Worker***. Методы CommitRecord() и AbortRecord() имеют один общий аргумент, который представляет собой запись, считанную из системного файла регистрации. По завершении транзакции именно из этой записи берется информация о модификации, которая должна быть либо зафиксирована, либо аннулирована, в зависимости от результатов "голосования" объектов, принимавших участие в операции. Этот процесс получил наименование *нормальной операции* (*Normal Operation*) CRM и схематически представлен на рис. 18.25.

На заметку

При возникновении в приложении фатальной ошибки файл системный регистрации будет содержать запись, которую сможет прочесть объект ***CRM Compensator***. Процесс считывания этой записи получил наименование *восстановления* (*recovery*). Здесь не будет рассматриваться процесс восстановления, поскольку необходимость в нем возникает только в исключительных ситуациях, когда приложение внезапно прекращает работать. Но учтите, что при использовании коммерческих программных продуктов об этом процессе ни в коем случае забывать нельзя.

Интерфейс IStorage в объекте *CRM Worker*

В интерфейсе IStorage нас интересуют методы Commit() и Revert(), с помощью которых транзакция в объекте хранения фиксируется или аннулируется. Эти операции можно выполнять только в отношении объекта хранения, который открыт в режиме, допускающем выполнение транзакций. В нашей программе оба составных файла открываются именно в этом режиме (вспомните о флаге STGM_TRANSACTED в реализации метода Swap() класса TStgSwapperImpl).

Сначала создадим COM-сервер CrmStg — вы уже знаете, что для этого нужно создать новый проект C++Builder типа Activex Library.

После этого можно приступать к разработке объекта ***CRM Worker***.

1. Создайте новый объект транзакций и дайте его сопряженному классу наименование **CrmStgWorker**. В списке Threading Model выберите вариант **Both**, а в списке Transaction Model — вариант **Requires a Transaction**.
2. Сформируйте объект автоматизации (Automation Object) и назовите его сопряженный класс **CrmStgCompensator**. Для этого объекта в списке Threading Model выберите вариант **Both**. К этому объекту мы вернемся позже, но его сопряженный класс нужно сформировать уже на данном этапе.
3. Выберите в левой панели окна редактора библиотеки типов TLE сопряженный класс CrmStgWorker, а в правой — откройте вкладку COM+. Установите для атрибута Call Synchronization значение **Required**. Также установите флажок Object Pooling.
4. Добавьте в интерфейс ICrmStgWorker следующий метод, который будет вызываться клиентом:

```
HRESULT _stdcall EnrollStgResource([in] IUnknown **pStorage );
```

На заметку

В данном случае используется интерфейс Iunknown, а не IStorage, поскольку ICrmStgWorker является дуальным интерфейсом.

5. Откройте файл CrmStgWorkerImpl.h, в котором содержится объявление класса TCrmStgWorkerImpl. Удалите из объявления класса член CComPtr<IObjectContext> m_spObjectContext. Нас совершенно не интересует контекст объекта ***CRM Worker***.

6. Добавьте в определение класса TCrmStgWorkerImpl член, в котором будет храниться ссылка на интерфейс ICrmLogControl объекта ***CRM Clerk***:

```
class ATL_NO_VTABLE TCrmStgWorkerImpl :
   public CComObjectRootEx<CComMultiThreadModel>,
   public CComCoClass<TCrmStgWorkerImpl,
          &CLSID_CrmStgWorker>,
   public IDispatchImpl<ICrmStgWorker,
          &IID_ICrmStgWorker,
          &LIBID_CrmStg>,
   public IObjectControl
{
   [ Пропущено... ]

   private:
   // Интерфейс ICrmLogControl объекта CRM clerk:
   CComPtr<ICrmLogControl> m_ptrICrmLogControl;
};
```

7. В методе Activate() организуйте создание объекта ***CRM Clerk*** и регистрацию объекта ***CRM Compensator***, который предполагается использовать (именно для этого мы ранее уже создали его сопряженный класс). Сопряженный класс объекта ***CRM Clerk*** называется CLSID_CRMClerk, а его структура определяется в <comsvcs.h>. Метод RegisterCompensator() получает в качестве первого аргумента идентификатор программы объекта ***CRM Compensator***. Второй аргумент — комментарий, а третий — флаг, который несет информацию о том, что объект ***CRM Compensator*** должен получать уведомления обо всех фазах выполнения транзакции. Программный код метода Activate() приведен в листинге 18.5.

Листинг 18.5. Реализация метода TCrmStgWorkerImpl::Activate()

```
STDMETHODIMP TCrmStgWorkerImpl::Activate(void)
{
   HRESULT hResult =S_OK;
   try
   {
      m_ptrICrmLogControl =NULL;
      // Создание объекта CRM clerk:
      HCHECK(m_ptrICrmLogControl.CoCreateInstance(
            CLSID_CRMClerk,
            NULL, CLSCTX_INPROC_SERVER));
      // Регистрация CRM Compensator.
      HCHECK(m_ptrICrmLogControl->RegisterCompensator(
            L"CrmStg.CrmStgCompensator",
            L"IStorage CRM compensator",
            CRMREGFLAG_ALLPHASES));
   }
   catch(ECOMError&e)
   {
      hResult = Error(e.Message, IID_IObjectControl,
               e.ErrorCode);
```

```
   }

   catch(...)
   {
      hResult =E_UNEXPECTED;
   }
   return hResult;
}
```

8. Освобождение интерфейса `ICrmLogControl` выполняется в методе `Deactivate()`:

```
STDMETHODIMP_(void)TCrmStgWorkerImpl::Deactivate(void)
{
   m_ptrICrmLogControl =NULL;
}
```

9. Реализация метода `EnrollStgResource()` представлена в листинге 18.6.

Листинг 18.6. Реализация метода `TCrmStgWorkerImpl::EnrollStgResource()`

```
STDMETHODIMP TCrmStgWorkerImpl::EnrollStgResource(
       LPUNKNOWN pStorage)
{
   HRESULT hResult = S_OK;
   try
   {
      // Проверить, выполнена ли ранее регистрация
      // соответствующего объекта CRM Compensator.
      ATLASSERT(m_ptrICrmLogControl != NULL);

      // Неявный вызов QueryInterface для интерфейса IStorage:
      CComQIPtr<IStorage> ptrStorage = pStorage;

      // Проверить, можно ли получить доступ к интерфейсу.
      BCHECK(ptrStorage != NULL, E_NOINTERFACE);

      // Сформировать псевдоним объекта, который будет
      // использоваться в процессе взаимодействия
      // CRM Worker и CRM Compensator.
      LPOLESTR pDisplayName = NULL;

      try
      {
         CComPtr<IBindCtx> pBC;
         HCHECK(CreateBindCtx(0,&pBC));
         CComPtr<IMoniker> ptrMoniker;
         HCHECK(CreateObjrefMoniker(ptrStorage,&ptrMoniker));
         ATLASSERT(ptrMoniker != NULL);
         HCHECK(ptrMoniker->GetDisplayName(
                   pBC,NULL,&pDisplayName));

         // Передать псевдоним объекту CRM Compensator
```

```
            // через файл регистрации CRM.
            BLOB blob;
            blob.pBlobData = reinterpret_cast<BYTE*> (pDisplayName);
            blob.cbSize =
               (wcslen(pDisplayName)+1)*sizeof(pDisplayName [0 ]);

            // Внести в файл регистрации запись "create file".
            HCHECK(m_ptrICrmLogControl->WriteLogRecord(
                                              &blob,1));

            HCHECK(m_ptrICrmLogControl->ForceLog());
         }

         __finally
         {
            if (pDisplayName != NULL)
               CoTaskMemFree(pDisplayName);
         }
      }
      catch(ECOMError&e)
      {
         hResult = Error(e.Message, IID_ICrmStgWorker, e.ErrorCode);
      }
      catch(...)
      {
         hResult =E_UNEXPECTED;
      }
      return hResult;
}
```

В методе `EnrollStgResource()` используется псевдоним `Objref`, который представляет объект хранения в файле регистрации. Для его формирования используется функция `CreateObjrefMoniker()`. Обратите внимание на то, что запись в файле регистрации представляет собой объект типа `BLOB`. Запись содержимого объекта в файл выполняется с помощью метода `m_ptrICrmLogControl->WriteLogRecord()`.

На заметку Более подробную информацию об использовании псевдонимов можно получить в справочной системе MSDN.

Созданный нами объект ***CRM Worker*** только передает ссылку на имя объекта хранения в соответствующий ему объект ***CRM Compensator***. Последний будет затем использовать это имя для обращения к интерфейсу `IStorage` объекта хранения и для вызова его методов `Revert()` или `Commit()`.

Разработка объекта *CRM Compensator*

При разработке объекта ***CRM Writer*** уже был создан (объявлен) сопряженный класс объекта ***CRM Compensator***. Но осталось сделать самое главное — вручную подготовить в составе этого сопряженного класса программный код реализации методов интерфейса

ICrmCompensator. В структуре этого интерфейса имеются десять методов, а сам интерфейс является непременным компонентом любого объекта ***CRM Compensator***, который создается в среде разработке на базе языка C++.

Смею надеяться, вам уже знакома методика реализации вручную методов COM-интерфейсов с помощью ATL, к тому же файлы реализации класса TCrmStgCompensatorImpl находятся на прилагаемом компакт-диске (это файлы CrmStgCompensatorImpl.h и CrmStgCompensatorImpl.cpp). Поэтому мы сосредоточимся только на реализации базовых методов интерфейса ICrmCompensator, а все прочие методы реализуются тривиально, и тот программный код, который вы найдете в файле CrmStgCompensatorImpl.cpp, по-моему, не требует комментариев.

Мы уже говорили выше о том, что метод CommitRecord() вызывается в том случае, когда все операции, связанные с транзакцией, выполнены успешно и нужно зафиксировать результат транзакции на физическом уровне. Программный код реализации метода TCrmStgCompensatorImpl::CommitRecord() представлен в листинге 18.7.

Листинг 18.7. Реализация метода TCrmStgCompensatorImpl::CommitRecord()

```
STDMETHODIMP TCrmStgCompensatorImpl::CommitRecord(
          CrmLogRecordRead crmLogRec,
          BOOL*pfForget)
{
   HRESULT hResult =S_OK;
   try
   {
      *pfForget =FALSE;
      // Извлечь имя объекта Objref, которое характеризует
      // объект хранения, вовлеченный в выполнение транзакции.
      WideString wstrDisplayName = reinterpret_cast<LPCOLESTR>
                         (crmLogRec.blobUserData.pBlobData);

      // Получить ссылку на интерфейс IStorage объекта хранения.
      CComPtr<IStorage> ptrStg;
      BIND_OPTS opts;
      opts.cbStruct = sizeof(BIND_OPTS);
      opts.dwTickCountDeadline = 1000;
      opts.grfMode = STGM_READWRITE;
      opts.grfFlags = 0;
      HCHECK(CoGetObject(wstrDisplayName, &opts, IID_IStorage,
                      reinterpret_cast<LPVOID*> (&ptrStg)));

      // Зафиксировать выполнение транзакции на физическом
      // уровне.
      HCHECK(ptrStg->Commit(STGC_DEFAULT));

      // "Забыть" об этой записи.
      *pfForget = TRUE;
   }

   catch(ECOMError&e)
   {
```

```
      hResult = Error(e.Message, IID_ICrmCompensator,
                      e.ErrorCode);
   }
   catch(...)
   {
      hResult = E_UNEXPECTED;
   }
   return hResult;
}
```

Как видите, частично это программа выполняет операции, обратные тем, которые выполнялись объектом ***CRM Worker***; аргумент `crmLogRec` содержит информацию, записанную в файл регистрации объектом ***CRM Worker***. Теперь эта информация считывается и интерпретируется. В результате формируется ссылка на объект хранения. Основную часть работы при этом выполняет функция `CoGetObject()`, возвращая ссылку на интерфейс `IStorage` объекта хранения, который характеризуется псевдонимом `Objref`.

Получив интерфейс `IStorage`, можно вызвать его метод `Commit()`, поскольку сам метод `CommitRecord()` вызывается только в том случае, если все операции, связанные с транзакцией, выполнены успешно.

Метод `AbortRecord()` вызывается в случае, когда одна из промежуточных стадий транзакции завершилась неудачей. В этом случае нужно восстановить исходное состояние объектов, вовлеченных в транзакцию. Реализация метода `AbortRecord()` во многом напоминает реализацию метода `CommitRecord()`. Единственная разница в том, что после получения ссылки на интерфейс `IStorage` объекта хранения, вместо его метода `Commit()`, вызывается метод `Revert()`:

```
// Аннулировать изменения
HCHECK(ptrStg->Revert());
```

Я рекомендую вам тщательно "проштудировать" программный код реализации класса `TCrmStgCompensatorImpl`, который находится в файлах на прилагаемом к книге компакт-диске, а сейчас можно сохранить и откомпилировать проект и перейти к следующему этапу — установке компонента CRM в приложение COM+.

Установка компонента CRM в приложение COM+

Для установки сервера `CrmStg` под управлением приложения `BCB Storage Objects` нужно выполнить те же операции, которые ранее выполнялись при установке новых компонентов в приложения COM+. После того как установлены объекты `CrmStg.CrmStgWorker` и `CrmStg.CrmStgCompensator`, в окне Component Services выделите `CrmStg.CrmStgWorker` и выберите в меню этого окна команду Action⇨Properties. Убедитесь, что атрибуты `CrmStgWorker` имеют следующие значения:

- `Transaction Support = Required`;
- `Enable Just In Time Activation = Checked`;
- `Synchronization Support = Required`;
- `Threading Model = Any Apartment` (это эквивалентно варианту `Both`).

Это именно те значения, которые были установлены при создании сервера `CrmStg`.

После этого щелкните на объекте `CrmStg.CrmStgCompensator` правой кнопкой мыши и выберите в контекстном меню команду Properties. Некоторые атрибуты этого компонента

нужно настроить с помощью диалогового окна Properties, независимо от того, какие настройки были выполнены в окне редактора библиотеки типов TLE на вкладке COM+. Убедитесь, что приведенные ниже атрибуты имеют перечисленные значения:

- `Transaction Support = Disabled`;
- `Enable Just In Time Activation = Unchecked`;
- `Synchronization Support = Disabled`;
- `Threading Model = Any Apartment` (это эквивалентно варианту `Both`).

Последний этап — разрешить работу компонентов CRM в нашем приложении.

1. В диалоговом окне Component Services выделите приложение `BCB Storage Objects` и выберите в меню команду Action⇨Properties. На экране откроется диалоговое окно Properties.
2. Откройте в нем вкладку Advanced и установите флажок Enable Compensating Resource Managers (Разрешить работу компонентов CRM) (рис. 18.26).

На заметку

Представленные в этой главе компоненты CRM имеют очень существенный, хотя на первый взгляд и незаметный, изъян. Предположим, что в операции обмена информацией участвуют два файла, значительно отличающиеся по длине. Если сначала будет выполнена фиксация транзакции по отношению к большему файлу и не хватит свободного места на носителе, то обращение к методу `CommitRecord()` компонента CRM закончится неудачей. При разработке компонентов CRM для коммерческих программных продуктов на такие "мелочи" нужно обращать внимание, поскольку в ситуациях, подобных описанной выше, они могут привести к полному краху созданного приложения.

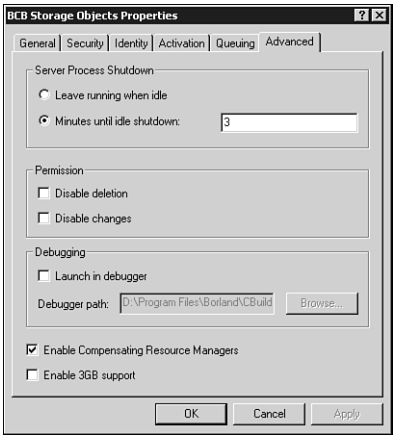


Рис. 18.26. Разрешение работы компонентов CRM в составе приложения COM+ `BCB Storage Objects`

Этим исчерпываются все "административные" настройки созданных CRM-компонентов сервера. Теперь вернемся к серверу `StgBusiness` и окончательно свяжем `CrmStgWorker` с объектом `StgSwapper`.

Новая версия объекта StgSwapper

Включение механизма CRM в объект StgSwapper выполняется достаточно просто.

1. Откройте проект StgBusiness в среде C++Builder.
2. Импортируйте библиотеку типов CrmStg — это выполняется по той же методике, которую мы уже неоднократно применяли при импортировании библиотек типов. Убедитесь. что файл CrmStg_TLB.cpp включен в состав проекта.
3. В начало файла StgSwapperImpl.h вставьте директиву включения файла CrmStg_TLB.H:

```
#ifndef StgSwapperImplH
#define StgSwapperImplH

#include "StgBusiness_TLB.h "

#include "CrmStg_TLB.H "

#include <comsvcs.h>
#define _MTX_NOFORCE_LIBS

#include <vcl\mtshlpr.h>

[ Пропущено... ]

class ATL_NO_VTABLE TStgSwapperImpl
   [ Пропущено... ]
{
   [ Пропущено... ]
};

[ Пропущено... ]
```

4. В качестве члена класса TStgSwapperImpl объявите объект CComPtr<ICrmStgWorker>:

```
[ Пропущено... ]
class ATL_NO_VTABLE TStgSwapperImpl
   [ Пропущено... ]
{
   [ Пропущено... ]
   private:
      CComPtr<IObjectContext> m_spObjectContext;
      CComPtr<ICrmStgWorker> m_ptrCrmWorker;
};
   [ Пропущено... ]
```

5. Откройте файл StgSwapperImpl.cpp. Измените программный код метода TStgSwapperImpl::Activate() и организуйте в этом методе формирование объекта ***CRM Worker***, прежде чем будет активизирован объект класса TStgSwapperImpl. В этом случае тоже рекомендуется использовать технологию работы со структурированными исключениями (листинг 18.8).

Листинг 18.8. Новая версия реализации метода `TStgSwapperImpl::Activate()`

```
STDMETHODIMP TStgSwapperImpl::Activate()
{
   HRESULT hResult =S_OK;
   try
   {
      // Получить контекст объекта:
      HCHECK(CoGetObjectContext(IID_IObjectContext,
             reinterpret_cast<void**> (&m_spObjectContext)));
      // Сформировать объект CRM Worker.
      HCHECK(m_ptrCrmWorker.CoCreateInstance(
             CLSID_CrmStgWorker));
   }
   catch(ECOMError &e)
   {
      hResult = Error(e.Message, IID_IObjectControl,
                e.ErrorCode);
   }
   catch(...)
   {
      hResult =E_UNEXPECTED;
   }
   return hResult;
};
```

6. Созданный объект ***CRM Worker*** разрушается при вызове метода `TStgSwapperImpl::Deactivate()`, программный код которого следует изменить:

```
STDMETHODIMP_(void)TStgSwapperImpl::Deactivate()
{
    m_ptrCrmWorker = NULL;
    m_spObjectContext.Release();
}
```

7. В метод `Swap()` добавьте два оператора регистрации интерфейсов `IStorage` обоих объектов составного файла в компонентах CRM (листинг 18.9). Эта операция выполняется обращением в методу `m_ptrCrmWorker->EnrollStgResource()`.

Листинг 18.9. Новая версия реализации метода `TStgSwapperImpl::Swap()`

```
STDMETHODIMP TStgSwapperImpl::Swap(BSTR File1,BSTR File2)
{
HRESULT hResult =S_OK;
   try
   {
      ATLASSERT(m_ptrCrmWorker != NULL);
      ATLASSERT(m_spObjectContext != NULL);

      // Открыть оба файла.
      CComPtr<IStorage> ptrSrc;
      HCHECK(StgOpenStorageEx(File1,
            STGM_READWRITE |STGM_TRANSACTED |
```

```
            STGM_SHARE_DENY_WRITE,
            STGFMT_ANY,
            0,NULL,NULL,
            IID_IStorage,
            reinterpret_cast<void**> (&ptrSrc)));

      CComPtr<IStorage> ptrDst;
      HCHECK(StgOpenStorageEx(File2,
            STGM_READWRITE | STGM_TRANSACTED |
            STGM_SHARE_DENY_WRITE,
            STGFMT_ANY,
            0,NULL,NULL,
            IID_IStorage,
            reinterpret_cast<void**> (&ptrDst)));

      // Регистрация двух объектов хранения.
      HCHECK(m_ptrCrmWorker->EnrollStgResource(ptrSrc));
      HCHECK(m_ptrCrmWorker->EnrollStgResource(ptrDst));

      // Сформировать временный файл.
      CComPtr<IStorage> ptrTmp;
      HCHECK(::StgCreateDocfile(NULL,
         STGM_DELETEONRELEASE | STGM_CREATE | STGM_READWRITE |
         STGM_TRANSACTED | STGM_SHARE_EXCLUSIVE,
         0, &ptrTmp));

      // Создать объекты Copier и Eraser.
      CComPtr<IStgCopier> Copier;
      HCHECK(Copier.CoCreateInstance(CLSID_StgCopier));
      CComPtr<IStgEraser> Eraser;
      HCHECK(Eraser.CoCreateInstance(CLSID_StgEraser));

      // Выполнить обмен информацией.
      HCHECK(Copier->Copy(ptrDst,ptrTmp));
      HCHECK(Eraser->Erase(ptrDst));
      HCHECK(Copier->Copy(ptrSrc,ptrDst));
      HCHECK(Eraser->Erase(ptrSrc));
      HCHECK(Copier->Copy(ptrTmp,ptrSrc));

      // Операция выполнена. "Голосуем" за фиксацию транзакции.
      m_spObjectContext->SetComplete();
   }

   catch(ECOMError &e)
   {
      // Возникла исключительная ситуация. Прервать операцию.
      m_spObjectContext->SetAbort();
      hResult =Error(e.Message,IID_IStgSwapper,e.ErrorCode);
   }

   catch(...)
```

```
   {
      // Возникла исключительная ситуация. Прервать операцию.
      m_spObjectContext->SetAbort();
      hResult =E_UNEXPECTED;
   }

   return hResult;
}
```

В этой версии вызовы `SetAbort()` и `SetComplete()` приведут в конце концов к вызову, соответственно, методов `AbortRecord()` и `CommitRecord()` объекта `CrmStgCompensator`.

На заметку

Учтите, что метод `CommitRecord()` физически фиксирует изменения в составном файле посредством интерфейса `IStorage`, который извлекается на основе информации, считанной из файла регистрации. А метод `AbortRecord()` аннулирует изменения.

Прелесть этого механизма в том, что он выполняет всю "черную" работу где-то там, "за сценой", а программисту не нужно ни о чем беспокоиться. Обратите внимание, что изменения совершенно не коснулись объектов `StgEraser` и `StgCopier`. Поскольку на них распространяется та же транзакция, которая сформирована в `StgSwapper`, в этих объектах не нужно непосредственно обращаться к ***CRM Worker***.

Создание клиентского приложения

Для проверки работоспособности разработанного сервера будут использоваться два файла, созданных программой MS Word. Мы должны убедиться, что сервер выполнит взаимный обмен содержимого обоих файлов (или не выполнит обмен). Имена этих файлов — `one.doc` и `two.doc` — оригинальностью не отличаются, но ведь и используются они для сугубо утилитарных целей. Процедура создания клиентского приложения, обращающегося к "услугам" сервера `StgBusiness`, достаточно очевидна.

1. Создайте заготовку приложения в среде C++Builder. Сохраните проект под именем `StgClient`, а созданный файл `Unit1.cpp` — под именем `Main.cpp`. Свойству `Name` экранной формы приложения присвойте значение **`MainForm`**.
2. Импортируйте библиотеку типов `StgBusiness` в проект. Затем в начало файла `Main.h` вставьте директиву включения файла `StgBusiness_TLB.h`:

   ```
   #include "StgBusiness_TLB.h"
   ```

3. В составе класса экранной формы `TMainForm` объявите объект класса `TCOMIStgSwapper`:

   ```
   class TMainForm :public TForm
   {
      __published: // Компоненты, включаемые C++Builder.

      private://Объявления, вносимые разработчиком вручную
         TCOMIStgSwapper FSwapper;
      public://Объявления, вносимые разработчиком вручную
         __fastcall TMainForm(TComponent*Owner);
   };
   ```

4. В класс экранной формы включите обработчик события `FormCreate()`, в котором будет создаваться объект `StgSwapper`:

```
void __fastcall TMainForm::FormCreate(TObject *Sender)
{
      FSwapper =CoStgSwapper::Create();
}
```

5. Добавьте в состав экранной формы кнопку (объект `btnSwap`) и реализуйте обработчик события `OnClick()` этой кнопки следующим образом:

```
void __fastcall TMainForm::btnSwapClick(TObject *Sender)
{
   WideString strExePath = ExtractFilePath(Application->ExeName);
   WideString strF1 = strExePath + L"one.doc";
   WideString strF2 = strExePath + L"two.doc";
   OLECHECK(FSwapper.Swap(strF1, strF2));
}
```

Сохраните проект и откомпилируйте его. Если компиляция пройдет успешно, это послужит лучшим доказательством того, что приложения на стороне клиента могут вообще не учитывать объекты транзакций и связанные с ними сложности. Обратите внимание: мы используем объект `StgSwapper` компонента COM+ не задумываясь о том, что он по самой своей природе является объектом транзакций. Единственное, что требуется от клиентского приложения, — сформировать экземпляр `StgSwapper` и работать с ним, как с любым другим COM-объектом. Помните, в самом начале главы я говорил о том, что COM+ очень облегчает жизнь программисту. Теперь, надеюсь, вы в этом убедились.

Резюме

В этой главе я старался ввести вас в круг наиболее важных, с моей точки зрения, концепций COM+. Хотя я кратко представил вам все службы COM+, основное внимание все же было уделено событиям и транзакциям, поскольку именно эти компоненты наиболее "активно" поддерживаются средствами C++Builder.

Надеюсь, после внимательного изучения этой главы вы сможете самостоятельно:

- создавать серверные приложения COM+ с помощью административных средств операционной системы;
- разрабатывать в среде C++Builder объекты событий COM+;
- формировать постоянные подписки на события COM+ с помощью службы ***Component Services*** операционной системы;
- программно формировать временную подписку на события COM+ с помощью служб каталога COM+;
- разрабатывать компоненты управления ресурсами CRM;
- разрабатывать объекты транзакций COM+.

Технология COM+ предлагает программисту много других возможностей, о которых мы не упоминали в этой главе. Материал, представленный в главе, следует рассматривать как первые шаги по длинной дороге освоения всех возможностей COM+.

Глава

19

Многоуровневые распределенные приложения на основе MIDAS 3

Боб Свот

Эта глава посвящена методике создания многоуровневых приложений, работающих с базами данных, (БД-приложений) на основе средств поддержки технологии Multi-Tier Distributed Application Services (MIDAS), реализованных в среде C++Builder.

Используя многоуровневую архитектуру при создании БД-приложений программист может таким образом распределить компоненты программы, что получит доступ к базе данных, размещенной на другом компьютере (сервере БД), не имея на клиентской машине полноценного набора средств работы с базами данных. Эта архитектура позволяет также сосредоточить все компоненты, определяющие логику работы с данными (так называемые *бизнес-правила*) в одном месте, а саму обработку распределить между компьютерами локальной сети.

Все примеры, приведенные в этой главе, ориентированы на версию MIDAS 3, которая входит в состав редакции *Enterprise edition* программного продукта C++Builder 5. Поэтому те читатели, которые не имеют доступа к C++Builder 5 Enterprise (хотя бы пробной версии), будут вынуждены при чтении этой главы ограничиться только теоретическим материалом.

Учтите, что перед началом практической работы с примерами, представленными на прилагаемом к книге компакт-диске, вам нужно внимательно прочесть инструкции в файле `README.TXT`, который находится в каталоге `MIDAS`.

Основы MIDAS

MIDAS поддерживает трехуровневую архитектуру БД-приложений, которая в классическом виде предполагает наличие трех основных компонентов:

- сервера базы данных, который размещается на одном компьютере;
- прикладного сервера на втором компьютере (компьютере среднего уровня);
- "тонкого" клиента, т.е. клиентского приложения с ограниченными ресурсами, на третьем компьютере.

В качестве сервера БД может выступать любая СУБД с достаточно широкими функциональными возможностями, например InterBase, Oracle, MS SQL Server и т.д. Прикладной сервер и клиентское приложение создаются в среде C++Builder. Прикладной сервер содержит средства поддержки бизнес-правил и средства манипулирования данными. На клиентские приложения возлагаются довольно ограниченные по объему и сложности задачи, связанные в основном с представлением данных конечному пользователю и приема от него команд.

"Отступлением" от классического варианта является компоновка системы, предполагающая совмещение на одном компьютере сервера БД и прикладного сервера. Но поскольку каждый из компонентов создается как автономный, и в этом случае можно считать архитектуру всей системы трехуровневой. Термин *N-уровневая организация вычислений* (*N-tier computing*) отражает тот факт, что все уровни можно разнести по отдельным компьютерам вычислительной сети. Например, сервер для учета персонала размещается на одном компьютере, а сервер для начисления зарплаты — на другом. Один из этих прикладных серверов может обращаться к серверу БД Oracle на третьем компьютере, а другой — к серверу InterBase на четвертом. В таком случае архитектура будет уже не трехуровневой, а *n*-уровневой.

На заметку

Строго говоря, термин *n-уровневый* в общем-то здесь не совсем к месту. Дело в том, что принадлежность к определенному уровню определяется не размещением задачи (программы, приложения) на определенном компьютере, а характером задачи. Поэтому описанный чуть выше вариант является все-таки трехуровневым — один уровень образуется двумя серверами БД, второй — двумя прикладными серверами, а третий — клиентским приложением. Схематически трехуровневая архитектура БД-приложения представлена на рис. 19.1.

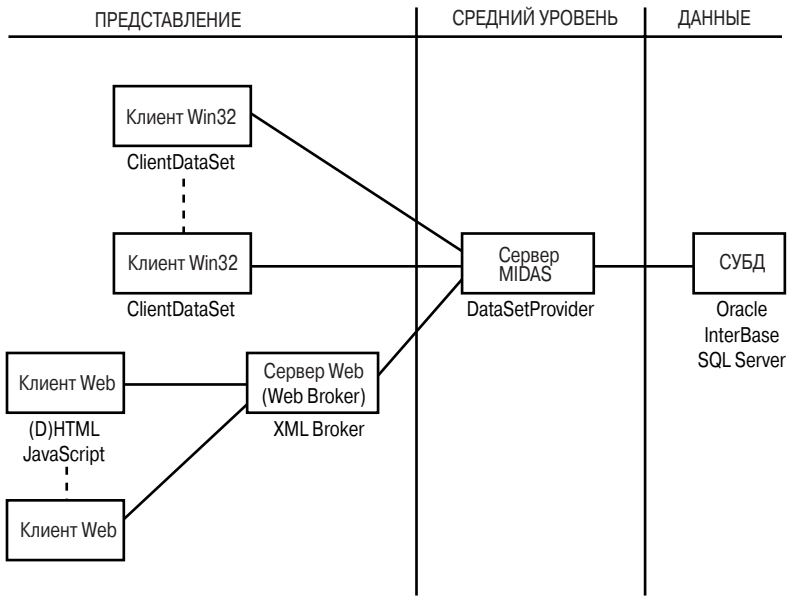


Рис. 19.1. Трехуровневая архитектура БД-приложения

MIDAS базируется на технологии, позволяющей упаковать наборы данных в отдельный пакет и пересылать их через локальную сеть в качестве аргументов удаленных вызовов определенных методов. Составным элементом этой технологии является преобразование набора данных в пакет типа `Variant` или `XML` на стороне сервера и последующая распаковка набора данных на стороне клиента и представление его пользователю в виде таблицы. Последняя операция выполняется с помощью компонентов `TClientDataSet` или `TMidasPageProducer`.

Можно рассматривать MIDAS и под другим углом зрения — это технология передачи набора данных из объектов `TTable` или `TQuery` на стороне сервера в объект `TClientDataSet` на стороне клиента. Компонент `TClientDataSet` внешне выглядит и функционирует точно так же, как `TTable` или `TQuery`, за исключением того, что не может быть подключен напрямую к BDE или любому другому драйверу СУБД — подключение возможно только через DLL-модуль MIDAS, который в таком случае является промежуточным звеном. В этом случае `TClientDataSet` получает необходимые ему данные после распаковки пакета типа `Variant`, переданного сервером.

MIDAS позволяет разработчику включать в программу на стороне клиента все стандартные компоненты C++Builder, в том числе и работающие с базами данных, но при этом клиентское приложение остается все-таки "тонким" клиентом, т.е. использует ограниченные вычислительные ресурсы. Это приложение не нуждается ни в каком ином драйвере БД, помимо самого модуля `MIDAS.DLL`. Установку `MIDAS.DLL` на стороне клиента принято называть *нульконфигурацией* (*zero-configuration*) тонкого клиента.

На заметку

Судя по названиям программных продуктов — Delphi, Kylix, а теперь MIDAS — в фирме Borland, очевидно, собрались большие любители греческой мифологии. Мидас (Midas) — греческий царь, которому бог Дионис даровал способность прикосновением руки обращать все предметы в золото. Но после того как в золото превратились все явства и все домочадцы, Мидас решил избавиться от слишком обременительного подарка и бросился мыть руки в реке. В результате весь песок в реке превратился в золотой, причем то, что по цвету он и сейчас выглядит, как золотой, подтверждают многочисленные туристы.

В самой технологии MIDAS можно выделить два уровня:

- Первый — компоненты, которые представлены на палитре компонентов ***Component Palette*** и входят в состав библиотеки VCL (`TDataSetProvider`, `TClientDataSet` и `TXMLBroker`). Первых два компонента находятся на вкладке Midas палитры компонентов, а последний — на вкладке InternetExpress.
- Второй — это протокол передачи сообщений по сети Internet. Этот уровень представлен протоколами DCOM, HTTP и "добрым старым" TCP/IP (сокетами). На вкладке Midas палитры компонентов размещаются различные компоненты соединений, которые ориентированы на работу с разными протоколами.

Компоненты, которые имеются в составе C++Builder, позволяют программисту с минимальными усилиями организовать соединение любой пары компьютеров в сети и передавать наборы данных между ними в обоих направлениях. В самом простом сценарии для создания прикладного сервера (среднего уровня БД-приложения) и клиентского приложения достаточно нескольких щелчков кнопкой мыши.

Использование MIDAS дает возможность разделить приложение на две части: одна реализует интерфейс конечного пользователя, а вторая — обработку данных и поддержку бизнес-правил. В "обычной" архитектуре клиент/сервер клиентское приложение, которое создается, например, с помощью C++Builder Professional, будет "толстым" (интеллектуальным) клиентом, поскольку в него нужно будет включить и средства подключения к серверу БД (обычно — СУБД). Многоуровневая архитектура позволяет использовать концепцию "тонкого" клиента, который не нуждается в собственных средствах подключения к серверу БД — эта задача возлагается на модуль `MIDAS.DLL`, Последний является, по сути, локальной "мини-СУБД". Сервер MIDAS взаимодействует с реальной СУБД точно так же, как "толстый" клиент в традиционной архитектуре клиент/сервер.

Клиенты и серверы MIDAS

Наиболее простой способ разобраться в том, как работает MIDAS, — создать приложение, включающее клиента и сервер. Лично я предпочитаю начинать с сервера MIDAS, который должен инкапсулировать и экспортировать набор данных. А следующий этап — создание клиента MIDAS, который подключается к серверу и отображает полученные от него данные в той или иной форме.

Создание сервера MIDAS

Ниже описан процесс создания простого сервера MIDAS. Программный код, который представлен в этом разделе, вы можете найти на прилагаемом к книге компакт-диске в папке `MIDAS\simple`. В этой папке вы найдете файл проекта `SimpleMidasServer.bpr` и все файлы программного кода.

Как обычно, все начинается с выбора в главном меню C++Builder команды File⇨New Application. В ответ C++Builder создаст заготовку приложения. Отображение экранной формы этого приложения на экране является подтверждением того, что приложение сервера загружено. Свойству `Caption` экранной формы присвоено значение **`C++Builder 5 Developer's Guide`**, но для того, чтобы ее была лучше видна на экране, я всегда включаю в нее надпись (элемент `TLabel`) с буквами покрупнее — задаю в свойстве `Font` какой-нибудь крупный и бросающийся в глава шрифт (например, `Comic Sans MS 24pt.`). Свойству `Caption` элемента `TLabel` я присваиваю имя сервера — в данном случае, **`My First MIDAS Server`**. Установите по своему вкусу размер экранной формы и цвет фона — свойство `Color`. Форма должна выглядеть примерно так, как на рис. 19.2.

Для того чтобы из обычного приложения сделать промежуточный сервер данных, нужно добавить в приложение модуль удаленных данных (Remote Data Module — в дальнейшем RDM-модуль). Этот специальный модуль данных можно найти в хранилище объектов ***Object Repository*** на вкладке Multi-Tier (рис. 19.3). Выберите в меню команду File⇨New и в диалоговом окне New Items откройте вкладку Multi-Tier.

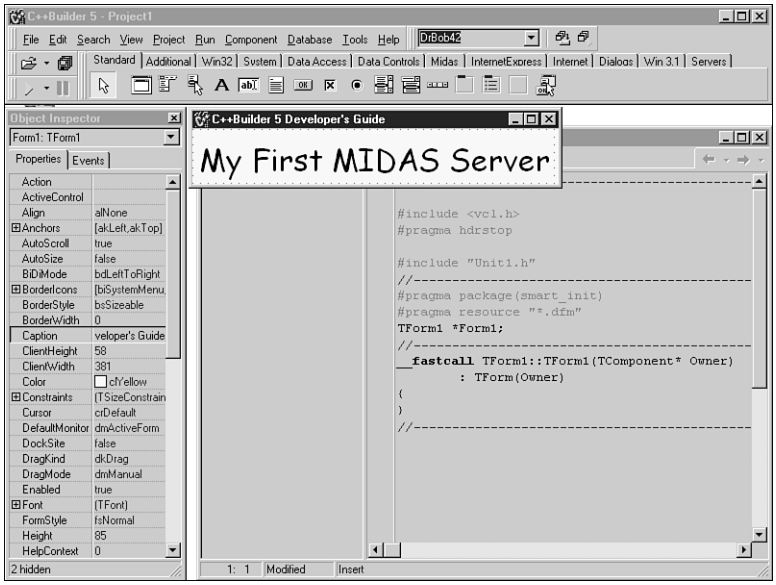


Рис. 19.2. Экранная форма приложения `My First MIDAS Server`

Из всего набора объектов на этой вкладке нас сейчас интересует только ярлык RDM-модуля Remote Data Module. Кроме него, на вкладке Multi-Tier представлены мастера CORBA (о них подробно будет рассказано в главе 20) и модуль транзакций Transactional Data Module. Последний можно использовать в сочетании с сервером ***Microsoft Transaction Server*** (MTS), но появление на рынке операционной системы Windows 2000 и модели COM+ практически свело на нет необходимость в этом компоненте.

После того как выбран ярлык Remote Data Module и вы щелкнули на кнопке OK, на экране открывается диалоговое окно New Remote Data Module Object (рис. 19.4).

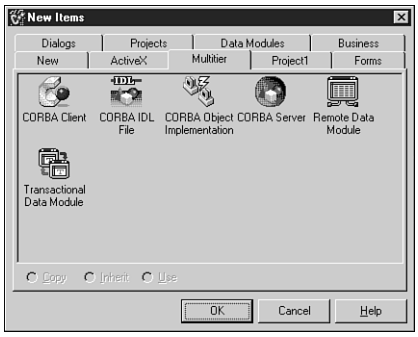


Рис. 19.3. Ярлык модуля Remote Data Module *в окне хранилища объектов*

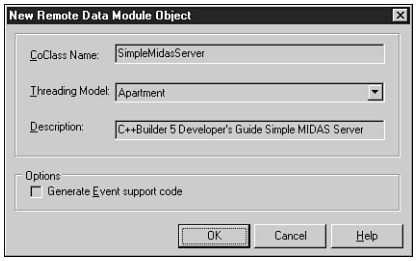


Рис. 19.4. Диалоговое окно New Remote Data Module Object

В поле CoClass Name этого окна нужно ввести имя внутреннего класса приложения. Имя этого класса нужно держать в памяти, а потому я предлагаю довольно запоминающееся **SimpleMidasServer** (имя должно быть представлено одним словом — пробелы не допускаются). В поле Threading Model (Модель потока) по умолчанию предлагается вариант **Apartment** (Секционированная), который практически всегда подходит. Другие варианты — Single (Однопотоковая), Free (Свободная), Both (Оба) и Neutral (Нейтральная). Хотя скорее всего вам никогда не придется изменять предлагаемый по умолчанию вариант, нужно все-таки иметь представление, что кроется за остальными.

- *Однопотоковая* (Single) модель в каждый момент времени поддерживает только один запрос со стороны клиента. Если поступят запросы одновременно от нескольких клиентов, то остальные будут ждать своей очереди. Таким образом, при использовании этой модели все ухищрения, связанные с параллельной отработкой запросов в многопотоковом режиме, оказываются ненужными, но такой вариант снижает производительность приложения при работе с большим количеством клиентов.
- *Секционированная* (Apartment) модель потоков позволяет обрабатывать одновременно несколько запросов клиентов. Создается несколько копий RDM-модуля, каждая из которых обрабатывает свой запрос. При этом формируется несколько потоков, и каждый запрос обрабатывается в своем потоке. Эту модель можно использовать при работе с наборами данных BDE, для чего вам понадобится компонент TSession, свойству AutoSessionName присвоено значение **True**.
- *Свободная* (Free) модель потоков предполагает, что в каждом потоке одновременно может обрабатываться несколько запросов клиентов. Поддерживать такую модель потоков довольно сложно и ее рекомендуется использовать при работе с наборами данных ADO.
- Вариант Both модели потока представляет собой по сути ту же свободную модель, в которую добавлена сериализация обратных вызовов интерфейсов клиентских приложений. Эту модель мы рассматривать не будем.
- Последний вариант — *нейтральная* (Neutral) модель. Это новый вариант, который применим только при работе с моделью COM+ в операционной системе Windows 2000, и представляет собой адаптацию секционированной модели к условиям функционирования COM+.

Ниже поля Threading Model размещается поле Description, в которое можно ввести краткое описание формируемого объекта. Введенный в это поле текст будет зафиксирован в системном реестре Windows и связан с ProgID интерфейса прикладного сервера. Этот же текст будет появляться и в окне реактора библиотеки типов при работе с интерфейсом сервера. В поле Description можно вводить любой текст — в данном случае я ввел **C++Builder 5 Developer's Guide Simple MIDAS Server**.

В самом низу диалогового окна находится флажок Generate Event Support Code (Формировать код поддержки обработки событий). Установка этого флажка приводит к тому, что мастер C++Builder формирует отдельный интерфейс управления событиями. Поскольку здесь не рассматривается методика обработки событий в объектах автоматизации, этот флажок устанавливать не следует.

После установки значений всех полей в диалоговом окне New Remote Data Module Object щелкните на кнопке OK — C++Builder сформирует RDM-модуль. Сформированный модуль очень похож на обычный модуль данных, по крайнем мере внешне. И работать с ним можно так же, как с обычным модулем данных — включить в его состав компонент TSession

(этот компонент находится на вкладке Data Access палитры компонентов) и присвоить свойству AutoSessionName компонента TSession значение **True**. (Это необходимо в том случае, если планируется использовать BDE и секционированную модель потоков.)

После включения в модуль компонента TSession можно добавить в него и другие компоненты из вкладки Data Access палитры. Например, можно включить компонент TTable и присвоить ему наименование **TableCustomer**. Свойству DatabaseName присвойте значение **BCDEMOS**. Откройте раскрывающийся список в поле значения свойства TableName и выберите в нем таблицу customer.db.

Предыдущие операции типичны для любого модуля данных. Теперь перейдем к тем операциям, которые связаны именно со спецификой этого модуля, работающего с *удаленными* данными. Откройте вкладку Midas палитры компонентов и отыщите компонент TDataSetProvider. Этот компонент играет ключевую роль в экспортировании наборов данных из модуля в окружающую среду (точнее, в клиентское приложение MIDAS). Включите компонент TDataSetProvider в состав модуля и присвойте его свойству DataSet значение **TableCustomer**. Это означает, что объект TDataSetProvider будет "провайдером" (распространителем, экспортером) таблицы TableCustomer. Эту таблицу он будет экспортировать в подключенное клиентское приложение MIDAS (как оно создается, будет описано в следующем разделе). Модуль RemoteDataModule сервера SimpleMidasServer должен выглядеть так, как показано на рис. 19.5.

На заметку

Ниже в этой главе мы более подробно рассмотрим структуру компонента TDataSetProvider, а сейчас обратите внимание на его свойство Exported. Этому свойству присвоено значение **true**, что означает — таблица TableCustomer экспортирована (по умолчанию). Если установить этому свойству значение **false**, то экспорт таблицы клиентам будет прекращен. Это бывает необходимо при круглосуточной работе сервера, когда время от времени выполняется резервное копирование определенных таблиц. После завершения операции резервного копирования свойству Exported вновь присваивается значение **true**.

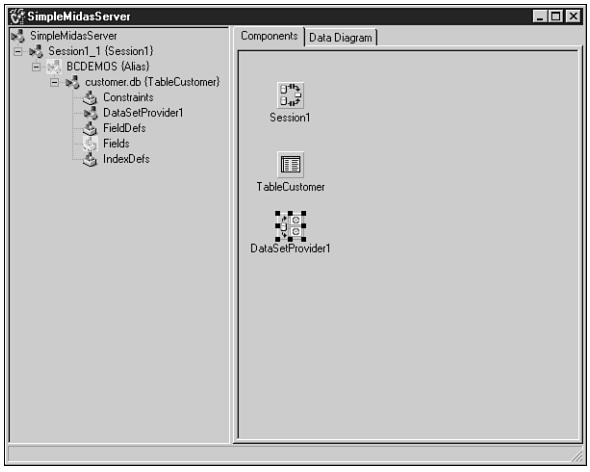


Рис. 19.5. RDM-модуль сервера SimpleMidasServer

На этом разработка простого сервера MIDAS завершается. Остается только сохранить файлы проекта (для этого следует вызвать команду Save All) и скомпилировать приложение.

В состав проекта входят файл главной экранной формы ServerMainForm.cpp, файл RDM-модуля SimpleMidasServerImpl.cpp и собственно файл проекта SimpleMidasServer.bpr. После первого запуска приложения на выполнение оно будет зарегистрировано в системном реестре. Теперь любой клиент MIDAS сможет отыскать этот сервер и подключиться к нему. Если вы перенесете файл приложения в другой каталог на том же компьютере, то сразу после переноса запустите приложение на выполнение — в системном реестре данные о местонахождении файла сервера MIDAS будут автоматически обновлены.

Ниже в этой главе будет рассказано и о том, как перенести сервер MIDAS на другой компьютер.

Регистрация сервера MIDAS

Теперь рассмотрим процедуру регистрации сервера MIDAS. Откройте файл SimpleMidasServerImpl.h. В этом файле найдите определение функции UpdateRegistry(). Для удобства работы с книгой мы продублировали этот фрагмент в листинге 19.1.

Листинг 19.1. Функция UpdateRegistry()

```
// Функция выполняет регистрацию объекта или
// удаление регистрационной записи объекта.
//
static HRESULT WINAPI UpdateRegistry(BOOL bRegister)
{
  TRemoteDataModuleRegistrar regObj(GetObjectCLSID(), GetProgID(),
         GetDescription());
  // Если не предполагается использовать объект для
  // подключения к Winsock или Web, сбросьте эти флаги.
  // Подробная информация о назначении других флагов
  // содержится в файлах atlmod.h и atlvcl.cpp.
  regObj.Singleton = false;
  regObj.EnableWeb = true;
  regObj.EnableSocket = true;
  return regObj.UpdateRegistry (bRegister);
}
```

Функция UpdateRegistry() обеспечивает автоматическую регистрацию RDM-модуля MIDAS 3 (или удаление регистрационной записи), если промежуточный прикладной сервер должен использоваться как сервер автоматизации. Обратите внимание на то, что функция UpdateRegistry() разрешает подключение к Winsocket и Web (используя протокол HTTP). Если необходимо заблокировать использование одного из этих протоколов, следует присвоить значение **false** членам EnableWeb или EnableSocket структурной переменной regObj.

В сопроводительной документации C++Builder 5 такая блокировка рассматривается как одна из мер обеспечения защиты приложения. Если сервер MIDAS не будет зарегистрирован в качестве средства, поддерживающего соединение Winsocket или Web, он станет "невидимкой" для всех компонентов, работающих через Winsocket или Web. Такая настройка отсутствовала в той версии сервера MIDAS, которую создавала среда C++Builder 4.

Вам не пришлось вручную вводить ни одной строки программного кода — все сделала среда C++Builder 5.

Создание клиента MIDAS

Клиентское приложение MIDAS может быть самым разным — обычной экранной формой, приложением типа *ActiveForm* или приложением Web-сервера (использующим Web Broker или InternetExpress). Фактически в качестве клиента MIDAS может работать приложение любого типа. Ниже будет показано, как в качестве клиента MIDAS использовать обычное приложение Windows. Этот клиент будет работать "в связке" с сервером MIDAS, который был рассмотрен в предыдущем разделе. Простое приложение, которое будет рассмотрено ниже, можно выполнять в режиме, когда и сервер, и клиент размещаются на одном и том же компьютере. В одном из последующих разделов будет описано, как распределить компоненты приложения между разными компьютерами локальной сети. В этом разделе мы будем ссылаться на программный код, который имеется в папке `MIDAS\simple` на прилагаемом к книге компакт-диске, в файлах проекта `SimpleMidasClient.bpr`.

Создайте заготовку нового приложения C++Builder — выберите в системном меню команду File⇨New Application.

На заметку

Сейчас можно было бы добавить в новое приложение модуль данных (выбрать команду File⇨New, а затем на вкладке New хранилища объектов — подходящий модуль данных). Чтобы уменьшить количество копий экрана при описании работы приложения, я решил не использовать модуль данных, а ограничиться невизуальными MIDAS-компонентами разработки.

Еще раз обращаю ваше внимание: в данном приложении клиента MIDAS модуль данных не используется. Вместо этого я включил все необходимые компоненты в состав главной экранной формы клиента MIDAS.

Но это не означает, что вы при желании не можете в таком клиентском приложении отделить обращение к данным от представления их конечному пользователю.

Проект, в котором демонстрируется использование модуля данных в клиентском приложении MIDAS, представлен в каталоге `MIDAS\simple2` на прилагаемом к книге компакт-диске. Файл этого проекта носит то же самое имя — `SimpleMidasClient.bpr`.

Прежде всего клиент MIDAS должен организовать соединение с приложением сервера MIDAS. Это соединение можно организовать на базе различных протоколов, например (D)COM, TCP/IP и HTTP. Соединение реализуют, соответственно, компоненты `TDCOMConnection`, `TSocketConnection` и `TWebConnection`. В первом приложении `SimpleMidasClient` будет использован компонент `TDCOMConnection`, поэтому перенесите его с вкладки MIDAS палитры компонентов на поле экранной формы создаваемого клиентского приложения MIDAS.

Компонент `TDCOMConnection` имеет свойство `ServerName`, где хранится имя сервера MIDAS, к которому нужно подключаться в процессе работы. В раскрывающемся списке возможных значений свойства `ServerName` в окне Object Inspector перечислены все зарегистрированные на компьютере серверы MIDAS 3. Если кроме вас никто на этом компьютере не работал, то скорее всего в этом списке значится только сервер `SimpleMidasServer.SimpleMidasServer`. Имя сервера состоит из двух компонентов: тот, что перед знаком точки, соответствует имени приложения, а тот, который после знака точки, — имени RDM-модуля. В данном случае и приложение, и RDM-модуль называются одинаково —`SimpleMidasServer`. После того как значение свойства `ServerName` задано, C++Builder самостоятельно установит и значение свойства `ServerGUID` компонента `TDCOMConnection`. Это значение будет извлечено из системного реестра. Если разработчик помнит значение идентификатора сервера, то можно сначала заполнить самостоятельно поле `ServerGUID`, а C++Builder автоматически установит соответствующее значение свойства `ServerName`.

Самое интересное начинается после двойного щелчка на имени свойства Connected компонента TDCOMConnection — выполняется переключение значения этого свойства с false на true. После этого C++Builder автоматически запустит сервер MIDAS, и будет организовано подключение. В результате на экране появится окно сервера SimpleMidasServer (рис. 19.6), процесс компоновки которого описан в предыдущем разделе.

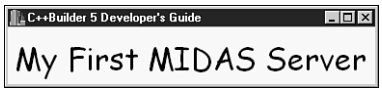


Рис. 19.6. Окно сервера SimpleMidasServer в процессе работы

На заметку

Закрыть MIDAS-сервер (остановить процесс выполнения) можно двумя способами: присвоить в окне Object Inspector свойству Connected компонента TDCOMConnection значение **false** или щелкнуть на кнопке закрытия окна сервера SimpleMidasServer. Первый способ сработает "чисто", а при попытке воспользоваться вторым — операционная система выведет окно с предупреждающим сообщением (рис. 19.7).

Если вы все-таки закроете приложение сервера MIDAS, то подключенный к нему клиент (точнее, компонент TDCOMConnection) "знать" об этом не будет и по-прежнему будет "считать" подключение активным. В реальных, а не учебных приложениях подобная ситуация может возникнуть, когда сервер MIDAS по каким-либо причинам прекратит работу. Подробнее о том, как справиться с нею, рассказано в разделе *Обработка ошибок* ниже в этой главе.

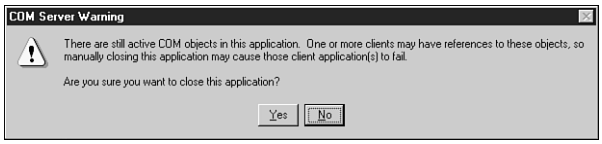


Рис. 19.7. Предупреждающее сообщение, которое формируется при попытке принудительно прервать работу сервера SimpleMidasServer

Снова дважды щелкните в окне Object Inspector на имени свойства Connected компонента TDCOMConnection — свойству будет присвоено значение **false**, а следовательно, соединение с сервером MIDAS будет разорвано, и сервер прекратит работу. Таким образом, манипулируя вручную значением свойства Connected, вы имели возможность убедиться, что соединение с сервером нормально работает. Теперь займемся импортированием набора данных, который экспортируется RDM-модулем сервера или, точнее, компонентом TDataSetProvider этого модуля. Перетащите на поле экранной формы клиентского приложения компонент TClientDataSet и свяжите его с компонентом TDCOMConnection посредством свойства RemoteServer. Компонент TClientDataSet будет получать данные от сервера MIDAS. Теперь нужно указать, какой провайдер будет использоваться для этого, другими словами, — из какого компонента TDataSetProvider желательно импортировать набор данных в компонент TClientDataSet. Для этого в структуре TClientDataSet предусмотрено свойство ProviderName. В окне Object Inspector откройте список в поле значения этого свойства — в перечень включены имена всех провайдеров, у которых свойство Exported имеет значение **true**. Боюсь, что на вашем компьютере оказался только один такой провайдер — компонент TDataSetProvider, который входит в состав сервера SimpleMidasServer. Он то нам и нужен.

После того как установлены значения свойств RemoteServer и ProviderName, нужно открыть (или активизировать) компонент TClientDataSet. Для этого в его структуре предусмотрено свойство Active, которому следует присвоить значение **true**. На этот раз сервер SimpleMidasServer

передает данные из таблицы TableCustomer через компонент TDataSetProvider и COM-соединение в компонент TDCOMConnection, который переадресует эти данные компоненту TClientDataSet клиентского приложения MIDAS.

На заметку

Перед установкой значения свойства ProviderName работа сервера SimpleMidasServer останавливается. Однако в список провайдеров включаются только те компоненты TDataSetProvider RDM-модулей, у которых свойство Exported имеет в текущий момент времени значение **true**. Поэтому, когда список открывается, сервер SimpleMidasServer вновь активизируется.

Теперь на поле экранной формы нужно перетащить компонент TDataSource, а затем открыть вкладку Data Controls палитры компонентов и включить в экранную форму визуальные компоненты, приспособленные для работы с данными. Поскольку мы не собираемся чрезмерно усложнять учебное приложение, ограничимся единственным компонентом TDBGrid. Свяжите свойство DataSet компонента TDataSource с TClientDataSet, а свойство DataSource компонента TDBGrid с TDataSource. Поскольку компонент TClientDataSet уже активизирован, вы сразу же увидите реальные данные, которые экспортировал сервер SimpleMidasServer.

На рис. 19.8 показано, как в этот момент выглядит экранная форма клиентского приложения SimpleMidasClient. Учтите, что предварительно я установил флажок Component Captions на вкладке Preferences диалогового окна Environment Options (оно открывается командой Tools⇨Environment Options).

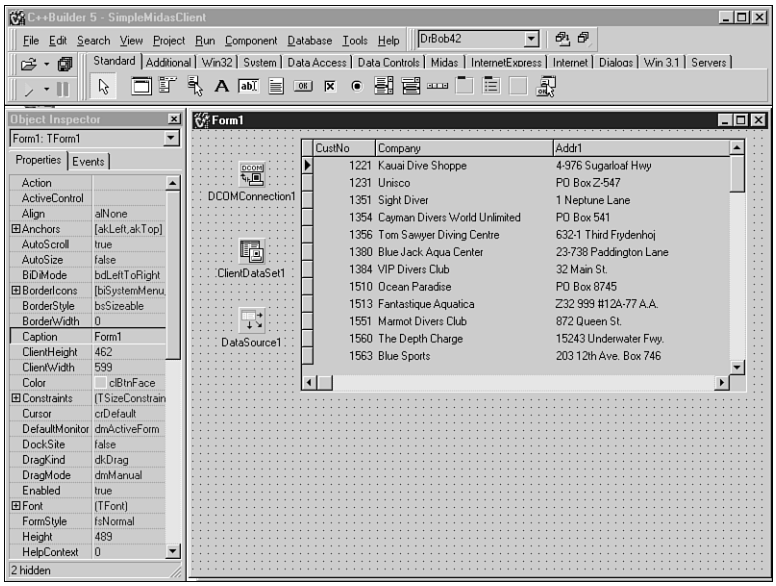


Рис. 19.8. Окно приложения SimpleMidasClient в процессе разработки

Вся основная работа практически завершена. Остались некоторые нюансы оформления. Установите заголовок экранной формы (свойству Caption объекта экранной формы присвойте значение, несущее осмысленную информацию о приложении, например **SimpleMidasClient**). Сохраните проект под именем SimpleMidasClient. После этого откомпилируйте его и запустите на выполнение. Опять обращаю ваше внимание на то, что в процессе разработки клиента MIDAS вам не понадобилось вводить вручную ни единой строки программного кода. Тем, кому это не по душе, могу обещать, что скоро все вернется "на круги своя".

Использование портфельной модели

После запуска клиента SimpleMidasClient в сетке экранной формы вы увидите данные из таблицы CustomerTable. Можно переходить от одной записи к другой, изменять значения полей и даже добавлять одни записи и удалять другие. Однако, после того как клиентское приложение будет закрыто, все внесенные изменения будут утеряны. Все внесенные изменения имеют отношение только к локальному объекту TClientDataSet и никак не затрагивают удаленную таблицу TableCustomer.

Возможности, с которыми вы сейчас столкнулись, характерны для так называемой "портфельной" (*briefcase*) модели БД-приложения (хотя и имеется один "маленький нюанс" — портфельная модель позволяет в дальнейшем внести изменения в удаленную базу данных). Работая с этой моделью приложения можно отключить клиента от сети и попрежнему работать с данными. Для этого набор данных, полученных от удаленной базы данных, сохраняется на локальном диске, после чего компьютер выключается и отсоединяется от сети. Затем данные можно снова загрузить с диска и работать с ними, не включаясь в сеть.

В дальнейшем, после включения в сеть, можно организовать соединение с "материнской" базой данных и обновить ее в соответствии с внесенными изменениями. Специальный механизм следит за корректностью изменений и уведомляет пользователя об обнаруженных ошибках или возникших конфликтах. Если, например, одна и та же запись отредактирована двумя пользователями, которые работали независимо друг от друга, нужно будет решить, какому из вариантов отдать приоритет.

Такая портфельная модель приложения идеально подходит для работы на портативных компьютерах, а также для разработки Web-страниц, когда желательно свести к минимуму объем информационного потока между Web-страницей и сервером БД.

Вы уже имели возможность убедиться, что разработанное приложение SimpleMidasClient работает только на локальном уровне с данными, поступившими в компонент TClientDataSet. Эти данные можно, конечно, сохранить на диске, а потом вновь загрузить. Для того чтобы организовать такое сохранение, включите в состав экранной формы кнопку (элемент TButton), присвойте ему наименование **ButtonSave**, а свойству Caption установите значение **Save**. Затем включите в модуль реализации экранной формы обработчик события OnClick этой кнопки:

```
void __fastcall TForm1::ButtonSaveClick(TObject *Sender)
{
  ClientDataSet1->SaveToFile("customer.cds", dfBinary);
}
```

При обращении к этой программе все записи из компонента TClientDataSet сохраняются в файле customer.cds, который помещается в текущий каталог. Расширение .cds означает ClientDataSet — *набор данных клиента*, но при желании вы можете выбрать любое понравившееся вам сочетание из трех букв в качестве расширения имени файла. Обратите внимание на флаг dfBinary, который передается в качестве второго аргумента при обращении к методу SaveToFile() класса TClientDataSet. Этот флаг означает, что данные будут сохранены в двоичном формате, который специфичен для продуктов Inprise/Borland. Можно выбрать и другие форматы. Например, передача флага dfXML приведет к сохранению данных в формате XML. В этом случае файл окажется более длинным (14,108 байт при объеме собственно данных в TableCustomer 7,493 байт). Правда, такое увеличение объема окупается тем, что с XML-форматом смогут работать и другие приложения. Поскольку в рассматриваемом примере мы не стремились к таким широким функциональным возможностям, я ограничился двоичным форматом.

По такой же схеме организуем и загрузку в компонент TClientDataSet данных из файла customer.cds. Включите в экранную форму еще одну кнопку (компонент TButton), установи-

те в свойстве Name значение **ButtonLoad**, а в свойстве Caption — значение **Load**. Текст обработчика события OnClick этой кнопки представлен ниже:

```
void __fastcall TForm1::ButtonLoadClick(TObject *Sender)
{
  ClientDataSet1->LoadFromFile("customer.cds");
}
```

Метод LoadFromFile() компонента TClientDataSet имеет только один аргумент и самостоятельно определяет формат считываемого файла.

Теперь модифицированное клиентское приложение может сохранять на локальном диске импортированные и отредактированные данные, а затем восстанавливать их при новом запуске.

Добавим в приложение еще одну кнопку, которая позволит управлять соединением с сервером MIDAS. По щелчку на этой кнопке будет переключаться значение свойства Active компонента TClientDataSet. Назовите эту кнопку **ButtonConnect**, а свойству Caption этого элемента управления присвойте значение **Connect**. Программный код обработчика события OnClick новой кнопки приведен в листинге 19.2.

Листинг 19.2. Обработчик события ButtonConnect OnClick

```
void __fastcall TForm1::ButtonConnectClick(TObject *Sender)
{
  if (ClientDataSet1->Active)
  {
   // Закрыть и отключить
    ClientDataSet1->Close();
    DCOMConnection1->Close();
  }
  else
  {
   // Открыть (соединение будет организовано автоматически)
//  DCOMConnection1->Open();
    ClientDataSet1->Open();
  }
}
```

На заметку Обращаю ваше внимание на то, что для разрыва соединения нужно закрыть компоненты TClientDataSet и TDCOMConnection, а для восстановления соединения нужно только открыть компонент TClientDataSet, который самостоятельно откроет TDCOMConnection.

Остается только проверить, подключены ли компоненты TDCOMConnection и TClientDataSet к серверу SimpleMidasServer. Эти компоненты *не должны* быть подключены, иначе при новой загрузке проекта SimpleMidasClient в C++Builder придется восстановить подключение — загрузить MIDAS-сервер SimpleMidasServer. Если по какой-либо причине сервер SimpleMidasServer не будет найден на компьютере, загрузить проект SimpleMidasClient в среду C++Builder не удастся. Поэтому я всегда рекомендую обеспечить разрыв соединения с сервером во время разработки приложения в среде C++Builder. Выполняется это очень просто: свойству Connected компонента TDCOMConnection присваивается значение **false** (это приводит к исчезновению с экрана окна сервера SimpleMidasServer). Такое же значение нужно присвоить и свойству Active компонента TClientDataSet (это при-

ведет к тому, что в экранной форме клиентского приложения данные в процессе разработки отображаться не будут).

На заметку

Если клиентское приложение пытается подключиться к серверу DCOM и не может найти его, система будет повторять попытки подключения еще некоторое время — примерно две минуты. В течение этих двух минут приложение окажется заблокированным. Если оно загружено в среду C++Builder, то заблокируется и C++Builder. На стадии разработки приложения эта проблема может быть решена единственным способом — нужно всегда перед сохранением файлов приложения устанавливать в свойстве `Connected` компонента `TDCOMConnection` значение **`false`**.

Теперь можно скомпилировать новую версию клиентского приложения `SimpleMidasClient` и запустить ее на выполнение. Сразу после запуска в экранной форме вы не увидите никаких данных (рис. 19.9). Щелкните на кнопке Connect — клиентское приложение подключится к серверу `SimpleMidasServer` и получит от него данные (эти данные сервер MIDAS, в свою очередь, затребовал у сервера БД). Но если у вас нет возможности подключиться к серверу `SimpleMidasServer` (например, приложение работает на портативном компьютере, который вы прихватили с собой на Гавайи), то последняя сохраненная версия данных может быть загружена с локального диска. Для этого нужно щелкнуть на кнопке Load. После редактирования измененные данные сохраняются на диске, если сеанс работы вы завершите щелчком на кнопке Save. Не забудьте также сохранить данные и в конце сеанса работы с сервером перед тем, как уезжать в командировку.

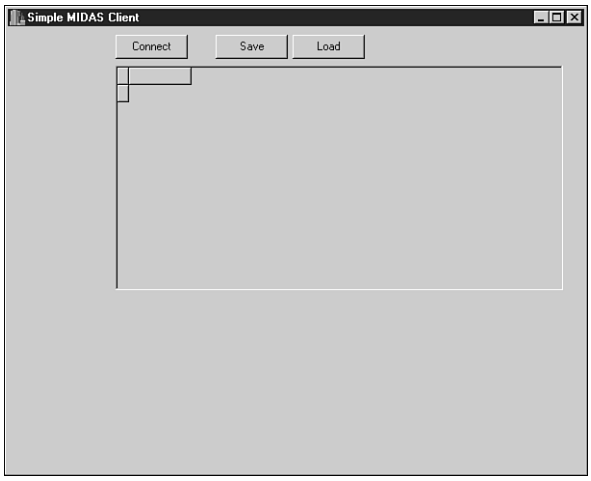


Рис. 19.9. Окно приложения `SimpleMidasClient`, когда приложение не подключено к серверу `SimpleMidasServer`

Можно продолжить совершенствование клиентского приложения — например, организовать блокировку кнопки Save до тех пор, пока в компонент `TClientDataSet` не будут загружены данные. В противном случае будет создан файл `customer.cds` нулевой длины. Если попытаться в дальнейшем загрузить этот файл, то по ряду причин компонент `TClientDataSet` будет самопроизвольно пытаться подключиться к серверу `SimpleMidasServer` (возможно это происходит потому, что сам компонент находится в активном состоянии, но не содержит ни единой записи данных). Конечно, это не очень приятная ситуация, которой следует избегать.

Использование метода `ApplyUpdates()`

Обновление данных в основной, удаленной базе данных выполняется с помощью метода `ApplyUpdates()` компонента `TClientDataSet`.

Перетащите на поле экранной формы клиентского приложения `SimpleMidasClient` четвертую кнопку, назовите этот элемент управления **`ButtonApplyUpdates`**, а его свойству `Caption` присвойте значение **`Apply Updates`**. Эту кнопку, как и кнопку Save, нужно разбло-

кировать только тогда, когда компонент TClientDataSet содержит какие-либо данные. (Я предоставляю вам возможность самостоятельно организовать блокировку. На всякий случай напоминаю, что вы всегда можете "подсмотреть", как это должно выглядеть в программе, обратившись к файлам проекта SimpleMidasClient на прилагаемом к книге компакт-диске. Этот проект находится в папке MIDAS\simple.)

Текст обработчика события OnClick кнопки ButtonApplyUpdates представлен ниже.

```
void __fastcall TForm1::ButtonApplyUpdatesClick(
          TObject *Sender)
{
  ClientDataSet1->ApplyUpdates(0);
}
```

Метод ApplyUpdates() компонента TClientDataSet принимает один аргумент — максимально допустимое количество ошибок перед тем, как процедура обновления будет прервана. Когда к серверу SimpleMidasServer подключено единственное клиентское приложение SimpleMidasClient, никаких проблем возникнуть не должно. Поэтому приложение SimpleMidasClient можно сразу же запустить на выполнение. Щелкните на кнопке Connect — программа организует соединение с сервером SimpleMidasServer и загрузит данные из удаленной таблицы. Затем с помощью кнопок Save и Load сохраните данные на локальном диске и вновь загрузите их в компонент TClientDataSet. Можете на время отключить свой компьютер от сети и редактировать данные в автономном режиме — именно так приходится работать с портфельной моделью приложения (роль портфеля играет портативный компьютер). Затем вновь подключите компьютер к сети, запустите приложение и щелкните на кнопке Apply Updates. После этого содержимое удаленной таблицы будет обновлено в соответствии с изменениями, внесенными в локальную копию набора данных.

Обработка ошибок

А что произойдет, если два клиентских приложения (оба — портфельные модели) подключатся к серверу SimpleMidasServer, импортируют один и тот же набор данных из таблицы TableCustomer, внесут в них изменения (каждое приложение — свои), причем в одну и ту же запись, например первую, и попытаются обновить содержимое основной удаленной базы данных. Клиентские приложения в том виде, как они существуют на данной стадии разработки, обратятся к методам ApplyUpdates() своих компонентов TClientDataSet и попытаются передать информацию об обновленном наборе данных в MIDAS-сервер. Если оба приложения передают методу ApplyUpdates() в качестве аргумента MaxErrors нулевое значение, то попытка обновления со стороны второго приложения будет прервана. Второе приложение могло бы передать отличное от нуля положительное значение аргумента в метод ApplyUpdates() и тем самым указать фиксированное количество ошибок или конфликтов обновления. Но даже если в качестве аргумента второе приложение передаст значение -1 (а при таком значении обновление будет продолжаться несмотря на обнаруженные ошибки и конфликты), те записи, которые были изменены предыдущим клиентским приложением, обновлены не будут. Придется прибегнуть к дополнительным "примирительным" операциям, чтобы выполнить новое обновление ранее обновленных записей или полей.

В составе C++Builder имеется специальное диалоговое окно, которое может помочь в этой ситуации. Если в разрабатываемом клиентском приложении MIDAS возможна описанная выше конфликтная ситуация, имеет смысл включить это окно в состав приложения. Это можно сделать следующим образом. Выберите в меню C++Builder команду File⇨New, откройте вкладку Dialogs хранилища объектов и выберите в ней ярлык Reconcile Error Dialog (рис. 19.10).

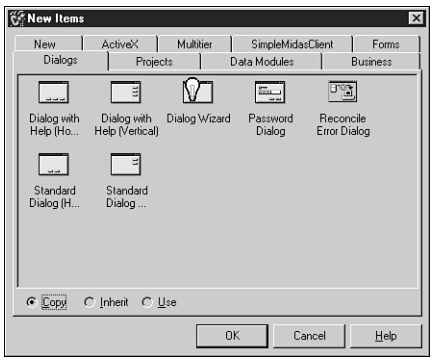


Рис. 19.10. Ярлык диалогового окна Reconcile Error Dialog *на вкладке* Dialogs *хранилища объектов*

После щелчка на кнопке OK в проект `SimpleMidasClient` будет добавлен новый модуль, который содержит программный код диалогового окна Update Error. С помощью этого окна пользователь может разрешить конфликты, возникающие при обновлении информации в базах данных (рис. 19.11).

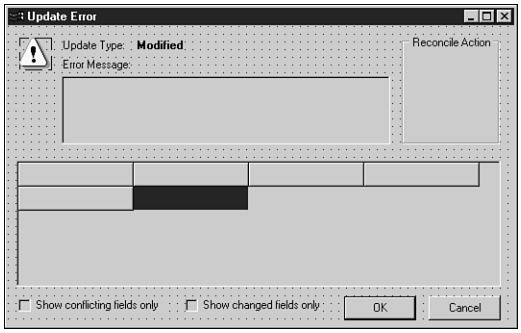


Рис. 19.11. Диалоговое окно Update Error *на стадии разработки*

Включив этот модуль в состав приложения `SimpleMidasProject`, нужно выполнить ряд операций. Во-первых, сохраните файлы проекта (присвойте файлу нового модуля имя **`ErrorDialog.cpp`**). Во-вторых, если ранее не был сброшен флажок автоматического создания форм на вкладке Preferences диалогового окна Environment Options (оно открывается с помощью команды Tools⇨Environment Options), проверьте, не включен ли автоматически компонент `TReconcileErrorForm` в состав приложения. На вкладке Forms диалогового окна Options (оно открывается командой Project⇨Options) находится список автоматически созданных форм и список доступных форм. В списке автоматически созданных форм не должен присутствовать элемент `ReconcileErrorForm`. Дело в том, что экранная форма `ReconcileErrorForm` будет создаваться динамически в процессе работы приложения, если в этом возникнет необходимость.

Как же использовать форму `ReconcileErrorForm`? В общем-то это не представляет особого труда. Если по каким-то причинам запись в удаленной таблице не была обновлена при вы-

полнении метода ApplyUpdates(), вызывается обработчик события OnReconcileError() компонента TClientDataSet. Заготовка обработчика события представлена ниже.

```
void __fastcall TForm1::ClientDataSet1ReconcileError(
      TClientDataSet *DataSet, EReconcileError *E,
      TUpdateKind UpdateKind,
      TReconcileAction &Action)
{

}
```

Обработчик принимает четыре аргумента:

- указатель на компонент TClientDataSet, который обнаружил ошибку;
- указатель на код ошибки ReconcileError, который содержит информацию о причинах, вызвавших данное событие;
- код типа операции UpdateKind, во время которой возникла ошибка (вставка, удаление или изменение);
- ссылка на значение Action, которая возвращается функцией, вызываемой обработчиком.

Action является значением перечислимого типа (типа enum):

- raSkip — запись обновлять не следует, но предлагаемые изменения должны быть зафиксированы в специальном журнале регистрации отвергнутых изменений; попытку обновления в дальнейшем можно будет повторить;
- raAbort — прервать процедуру разрешения конфликтов; после этого прекращается генерирование событий для обработчика OnReconcileError() при обнаружении ошибок;
- raMerge — слияние обновленной записи с текущей записью в удаленной базе данных; при этом изменяются только те поля удаленной записи, которые были изменены в клиентском приложении;
- raCorrect — обновленная запись заменяется скорректированными данными, которые формируются в обработчике события (или внутри формы ReconcileErrorDialog); в этом случае необходимо вмешательство пользователя;
- raCancel — выполняется откат всех изменений в текущей записи, и она возвращается к тому виду, который имела на локальном компьютере;
- raRefresh — выполняется откат всех изменений, и в запись возвращаются все данные из текущей (удаленной) базы данных.

Задача программиста — передать аргументы вызова OnReconcileError() в функцию HandleReconcileError() окна ErrorDialog.

Для этого нужно выполнить две операции. Сначала вставьте в файл экранной формы приложения SimpleMidasClient директиву включения файла заголовка модуля ErrorDialog. Щелкните на ClientMainForm и выберите команду File⇨Include Unit Hdr — на экране откроется диалоговое окно Use Unit (рис. 19.12).

Поскольку модуль ClientMainForm является текущим, то в списке диалогового окна Use Unit будет присутствовать только один элемент — ErrorDialog. Выделите его и щелкните на кнопке OK.

Далее в обработчик события OnReconcileError компонента TClientDataSet нужно вставить оператор вызова функции HandleReconcileError() модуля ErrorDialog (того самого, который только что был включен в список импорта модуля ClientMainForm).

Функция HandleReconcileError() принимает те же четыре аргумента, что и обработчик события OnReconcileError(), поэтому ваша задача — продублировать их в операторе вызова HandleReconcileError() (листинг 19.3).

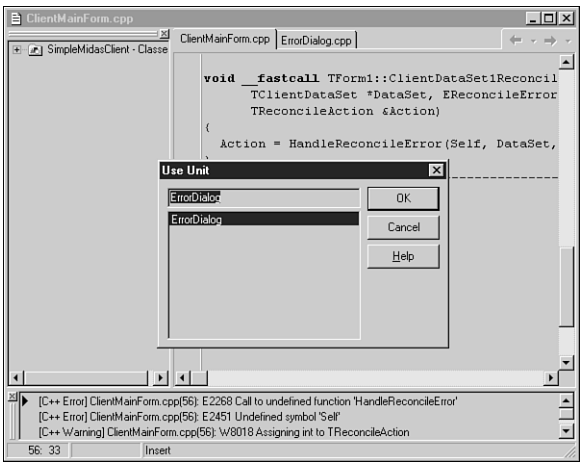


Рис. 19.12. Вставка в модуль ClientMainForm директивы включения файла заголовка ErrorDialog.h

Листинг 19.3. Обработчик события OnReconcileError()

```
void __fastcall TForm1::ClientDataSet1ReconcileError(
      TClientDataSet *DataSet, EReconcileError *E,
      TUpdateKind UpdateKind,
      TReconcileAction &Action)
{
  Action = HandleReconcileError(this, DataSet, UpdateKind, E);
}
```

Разрешение конфликтов в работающем приложении

Теперь посмотрим как выглядит на практике процедура разрешения конфликтов. Для этого нам потребуется два (или более) клиентских приложения SimpleMidasClient, работающих одновременно. Выполните следующие операции.

1. Запустите первое приложение SimpleMidasClient и щелкните на кнопке Connect (после этого будет загружен и сервер SimpleMidasServer).
2. Запустите второе приложение SimpleMidasClient и в его окне также щелкните на кнопке Connect. Ранее запущенный сервер SimpleMidasServer загрузит данные в это приложение, и вы увидите их в его окне.
3. В окне первого приложения SimpleMidasClient измените значение в поле Company первой записи.
4. Измените это же поле в окне второго приложения SimpleMidasClient, но, естественно, не так, как в первом приложении.

5. Щелкните на кнопке Apply Updates в окне первого приложения `SimpleMidasClient`. Внесенное изменение зафиксируется в удаленной базе данных, и при этом не возникнет никаких конфликтов.
6. Щелкните на кнопке Apply Updates в окне второго приложения `SimpleMidasClient`. На этот раз должна возникнуть конфликтная ситуация, поскольку первое приложение уже обновило по-своему поле Company в первой записи набора данных. В результате будет вызван обработчик события `OnReconcileError()`.
7. На экране отобразится диалоговое окно Update Error (рис. 19.13), и у вас появится возможность поэкспериментировать с переключателем Reconcile Actions. Этот переключатель имеет шесть состояний: Abort (Прервать), Skip (Пропустить), Cancel (Отказ), Correct (Корректировать), Refresh (Обновить) и Merge (Слить). Обратите особое внимание на отличие между операциями Skip и Cancel, а также на отличия между операциями Correct, Refresh и Merge.

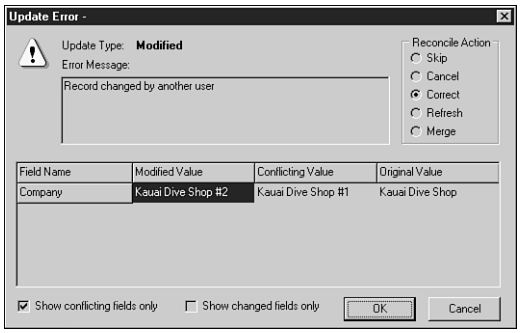


Рис. 19.13. Диалоговое окно Update Error в процессе работы клиентских приложений

При выполнении операции Skip происходит переход к следующей записи, причем текущая не обновляется (по крайней мере в момент выполнения операции). Отвергнутые изменения фиксируются в специальном журнале регистрации. При выполнении операции Cancel не только пропускается обновление текущей записи, но и прекращается весь процесс обновления, т.е. фактически прерывается выполнение метода `ApplyUpdates()`.

При выполнении операции Refresh восстанавливается исходная информация в обновленной записи, причем данные извлекаются из сервера. Таким образом отвергаются изменения, внесенные в запись не только данным приложением, но и его "конкурентом" (чтобы никому не было обидно). При выполнении операции Merge данные, полученные от текущего приложения, сливаются с данными в этой же записи, полученными *от сервера*, т.е. изменения вносятся не в тот вариант записи, который сформирован первым клиентским приложением, а в исходные данные, полученные от удаленной базы.

Наиболее "мощной" операцией является Correct, поскольку она предоставляет пользователю возможность откорректировать конфликтующую запись непосредственно в диалоговом окне обработчика событий.

Удаленный доступ к серверу

Сейчас вы должны иметь уже достаточно отчетливое представление о механизмах работы сервера `SimpleMidasServer` и его клиентов. Но до сих пор оба приложения выполнялись на одном и том же компьютере. В этом разделе мы рассмотрим, как организуется взаимодействие клиента с удаленным сервером MIDAS.

При организации распределенных DCOM-приложений настоятельно рекомендуется размещать серверный DCOM-компонент на том компьютере, где установлен сервер домена Windows NT. Клиентские приложения желательно размещать на компьютерах того же домена локальной сети. Если в вашем распоряжении нет сервера домена NT, то желательно, чтобы компьютер сервера и клиента имели одинаковые системные имена и пароли, по крайней мере на время отладки и тестирования распределенного приложения. В комплект операционной системы Windows 98 входят средства поддержки DCOM, а если вы пользуетесь операционной системой Windows 95, придется самостоятельно установить эти средства на компьютер. Необходимые модули DLL можно перегрузить с Web-сервера фирмы Microsoft.

Серверное приложение должно быть зарегистрировано и на собственном компьютере, и на компьютерах клиентов. Клиентское приложение сможет и самостоятельно отыскать нужный ему сервер и запустить его, даже если вы не зарегистрировали предварительно сервер на компьютере клиента, но средства поддержки COM не смогут организовать маршалинг данных, если не будут располагать в системном реестре компьютера клиента данными о библиотеке типов сервера. Регистрация сервера выполняется однократным запуском его на соответствующих компьютерах. Можно поступить и по-другому — запустить сервер на том компьютере, где он и размещается, а на компьютерах клиентов зарегистрировать файлы библиотеки типов (с расширением .TLB) с помощью утилиты `TRegSvr.exe` (ее вы сможете отыскать в папке `CBuilder\bin`). Файл библиотеки типов разработанного ранее в этой главе приложения сервера называется `SimpleMidasServer.tlb`. Он формируется автоматически средой C++Builder при создании приложения сервера.

Для того чтобы проверить, как работает удаленный доступ к серверу, нужно только скопировать выполняемый файл клиентского приложения на другой компьютер (если, конечно, все операции, связанные с регистрацией сервера на этом компьютере, уже выполнены). Кроме того, на этом компьютере должен быть установлен DLL-модуль `MIDAS.DLL`.

Создание сервера MIDAS типа „главный–подчиненный“

В этом и последующих разделах мы будем рассматривать более сложное MIDAS-приложение. Сначала речь пойдет об усложненном варианте MIDAS-сервера. Программный код этого приложения вы сможете найти на прилагаемом к книге компакт-диске в папке `MIDAS`; файл проекта носит наименование `MidasServer.bpr`.

Для нового приложения организуйте отдельный каталог. Приложение сервера создается в среде C++Builder следующим образом.

- Выполните команду File⇨New Application — C++Builder создаст заготовку нового приложения. Файлу экранной формы приложения присвойте имя **`MainForm.cpp`**, а файлу проекта — **`MidasServer.bpr`**.
- Как и в случае с первым приложением (`SimpleMidasServer`), в экранной форме скомпонуйте надпись, которая позволит при запуске сервера однозначно идентифицировать это окно на экране (рис. 19.14). Для этого в экранную форму нужно добавить элемент управления типа `TLabel` или какое-нибудь понравившееся вам изображение.
- Далее по методике, описанной в одном из предыдущих разделов, запустите мастер ***Remote Data Module Wizard*** (его ярлык находится на вкладке Multi-Tier диалогового окна хранилища объектов New Items). На этот раз в поле CoClass Name задайте имя сопряженного класса **`CustomerOrders`**, а в других элементах управления оставьте значения, предлагаемые мастером по умолчанию. В результате будет создан MIDAS-сервер `MidasServer.CustomerOrders` — это имя вы в дальнейшем увидите в окне Object Inspector при создании клиентского приложения.

- На поле созданного RDM-модуля перетащите два компонента TTable. Одному из них присвойте имя **TableCustomer**, а другому — **TableOrders**.
- Свойству DatabaseName каждого из компонентов TTable присвойте значение **BCDEMOS**.

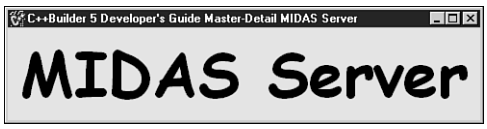


Рис. 19.14. Экранная форма сервера MIDAS типа "главный–подчиненный"

- Выделите компонент TableCustomer и выберите **customer.db** в качестве значения его свойства TableName. Таким же способом установите значение **orders.db** в свойстве TableName компонента TableOrders.

Теперь нужно сформировать отношение "главный–подчиненный" между компонентами TableCustomer и TableOrders.

Откройте вкладку Data Diagram RDM-модуля CustomerOrders (эта вкладка — новинка версии C++Builder 5). На этой вкладке можно графическими средствами сформировать отношение "главный–подчиненный" между компонентами типа TTable, которые входят в состав RDM-модуля.

- Сначала нужно перетащить таблицы customer.db и orders.db с левой панели (ClassExplorer) в правую (Data Diagram).
- К сожалению, после этой операции поля таблиц в правой панели отображены не будут. Придется снова выделить в левой панели таблицы, открыть узлы customer.db и orders.db, выбрать узлы Fields каждой из таблиц и поочередно щелкнуть на них правой кнопкой мыши. После каждого щелчка выбирайте в контекстном меню команду Add All Fields. В результате изображение на экране должно походить на то, что представлено на рис. 19.15.

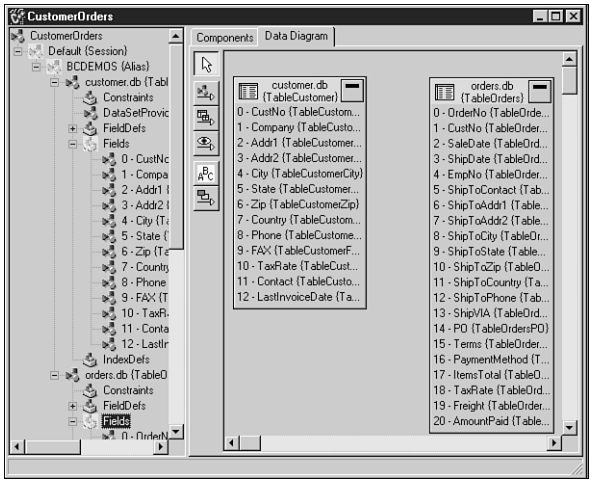


Рис. 19.15. На вкладке Data Diagram представлена структура таблиц компонентов TableCustomer и TableOrders

- Далее нужно сформировать связь между полями двух компонентов. Эта связь и определит отношение “главный–подчиненный”. Щелкните на третьей пиктограмме сверху на вкладке Data Diagram. Затем щелкните на поле того компонента, который будет играть роль главной таблицы в этой связи (компонента `TableCustomer`) и протяните указатель к тому компоненту, который будет играть роль подчиненной таблицы (`TableOrders`). После того как кнопка мыши будет отпущена, на экране появится диалоговое окно Field Link Designer, В этом диалоговом окне нужно указать, между какими полями таблиц устанавливается связь. В данном случае сначала выберите **`CustNo`** в поле Available Indexes, а затем — элементы **`CustNo`** в списках Detail Fields и Master Fields. После щелчка на кнопке Add будет сформирована связь между полями `CustNo` в обоих таблицах (рис. 19.16).

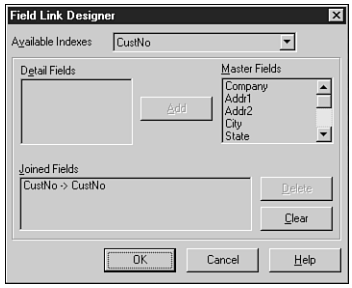


Рис. 19.16. Диалоговое окно Field Link Designer, в котором формируется связь `CustNo->CustNo`

После щелчка на кнопке OK будет сформировано отношение “главный–подчиненный” между компонентами `TableCustomer` и `TableOrders`. Графически это отношение будет представлено на вкладке Data Diagram (рис. 19.17.

Теперь нужно переключиться на вкладку Components RDM-модуля `CustomerOrders`. Здесь мы организуем экспорт таблиц клиентам.

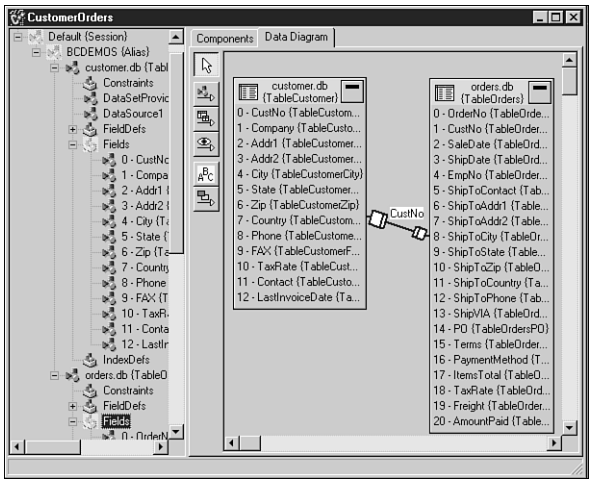


Рис. 19.17. Отношение “главный–подчиненный”, представленное в графической форме на вкладке Data Diagram

На заметку

На вкладке Components вы увидите компонент `TDataSource`, которого ранее не было. Этот компонент сформирован средой C++Builder после того, как на вкладке Data Diagram было сформировано отношение “главный–подчиненный” между двумя компонентами. Свойство `DataSet` компонента `TDataSource` указывает на `TableCustomer`, свойство `MasterSource` компонента `TableOrders` указывает на компонент `TDataSource`, а свойства `MasterFields` и `IndexName` компонента

TableOrders указывают на поле CustNo. Конечно, все это можно было бы задать и самостоятельно вручную, но значительно проще воспользоваться услугами среды разработки C++Builder.

Экспортирование наборов данных „главный–подчиненный“

В приложении сервера SimpleMidasServer для экспортирования таблицы TableCustomer из RDM-модуля использовался единственный компонент TDataSetProvider. Но в данном случае нам понадобится и два компонента TDataSetProvider: один будет экспортировать из RDM-модуля таблицу TableCustomer, а второй — TableOrders. Сами по себе эти компоненты смогут экспортировать только эти две таблицы, но не отношение “главный–подчиненный” между ними. Придется на стороне клиента воспроизвести всю процедуру формирования отношения. В обычном приложении можно было бы согласиться с таким подходом (определять отношение между таблицами на стороне клиента), но в многоуровневом приложении в такой ситуации возникают две проблемы.

Во-первых, компонент TClientDataSet клиентского приложения MidasClient должен заполнить и сохранить всю информацию из записей подчиненной таблицы, получив ее от сервера, даже в том случае, если клиенту после формирования отношения “главный–подчиненный” необходимы из нее только несколько записей. Таким образом, значительный объем информации передается без всякой пользы, загружая сеть. Конечно, с этой проблемой можно справиться, если передавать от клиента серверу какие-либо параметры, позволяющие фильтровать записи перед передачей, но это требует дополнительной работы, причем отлаживать такую программу очень сложно.

Вторая проблема, которая возникает при организации отношения “главный–подчиненный” на стороне клиента, касается процедуры обновления данных в связанных таблицах. Она возникает вследствие того, что компонент TClientData не может самостоятельно организовать совместное обновление нескольких таблиц в процессе выполнения единственной транзакции.

В результате приходим к выводу, что не следует экспортировать наборы данных, связанные отношением “главный–подчиненный”, как отдельные объекты. К счастью, компонент TDataSetProvider располагает средствами для экспортирования двух или более таблиц, связанных таким образом, как единого объекта. Для этого нужно подключить TDataSetProvider к тому компоненту TTable, который представляет главную таблицу на стороне сервера MIDAS, в данном случае — таблицу TableCustomer. Весь фокус в том, что главная таблица будет автоматически включать DataSetField для записей подчиненной таблицы, причем только для тех, которые связаны с текущей записью главной таблицы, и только эти записи будут передаваться по сети.

Из всего сказанного следует, что вам нужно перетащить один компонент TDataSetProvider на вкладку Components RDM-модуля и подключить его свойство DataSet к таблице TableCustomer (компонент TDataSetProvider можно найти на вкладке Midas палитры компонентов). Именно новый компонент TDataSetProvider и будет экспортировать данные из обоих таблиц RDM-модуля.

Сохраните файл RDM-модуля после внесения этих изменений (присвойте ему имя CustomerOrdersImpl.cpp). Опять обращаю ваше внимание на то, что в процессе разработки нового RDM-модуля вам не пришлось ввести вручную ни единой строки программного кода. Скомпилируйте проект MidasServer и запустите его, чтобы зарегистрировать в системном

реестре компьютера. Далее нам потребуется разработать клиентское приложение, которое будет пользоваться услугами этого сервера MIDAS.

Создание клиента MIDAS, работающего со связанными таблицами

Файлы нового клиентского приложения MIDAS, работающего с сервером `MidasServer`, находятся на прилагаемом к книге компакт-диске, в папке `MIDAS`. Имя файла проекта — `MidasClient.bpr`. Но ниже будет показано, как самостоятельно создать такое приложение.

Начните новое приложение C++Builder — выберите в главном меню команду File⇨New Application. Сохраните файл главной экранной формы приложения под именем `ClientMainForm.cpp`, а файл проекта — под именем `MidasClient.bpr`. Перетащите на поле экранной формы компонент `TDCOMConnection` (он находится на вкладке Midas палитры компонентов). В раскрывающемся списке возможных значений свойства `ServerName` компонента `TDCOMConnection` в окне Object Inspector вы увидите элементы `SimpleMidasServer.SimpleMidasServer` (это первый из разработанных нами серверов) и `MidasServer.CustomerOrders` (это второй вариант сервера). Естественно, что в качестве значения свойства `ServerName` следует выбрать второй элемент, **`MidasServer.CustomerOrders`**. Можно установить значение **`true`** для свойства `Connected` компонента `TDCOMConnection` и проверить, правильно ли загружается сервер `MidasServer`.

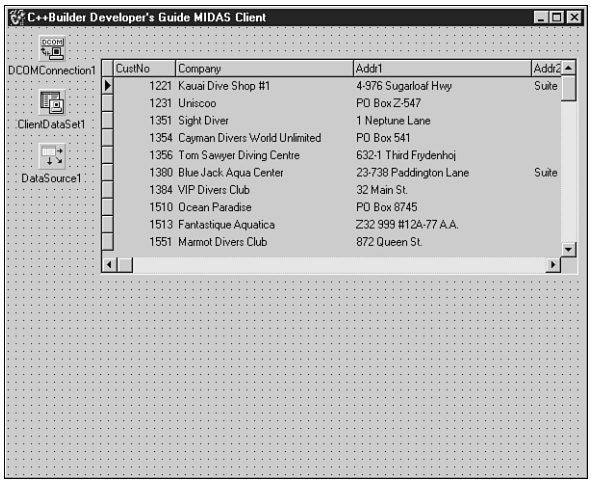


Рис. 19.18. Главная форма клиентского приложения MIDAS на стадии разработки

Теперь перетащите компонент `TClientDataSet` на поле экранной формы (он находится на той же вкладке Midas). Этот компонент будет через `TDCOMConnection` извлекать данные из RDM-модуля. Свяжите свойство `RemoteServer` компонента `TClientDataSet` с компонентом `TDCOMConnection`, которому по умолчанию присвоено имя `DCOMConnection1`. Далее нужно будет выбрать провайдер, который экспортирует данные из RDM-модуля. В рассматриваемой ситуации выбор невелик — только один провайдер (экспортируется только таблица `TableCustomer` с помощью `DataSetProvider1`). Выберите этот элемент в качестве значения свойства `ProviderName` компонента `TClientDataSet` (по умолчанию ему будет присвоено имя `DataSetProvider1`).

Теперь откройте вкладку Data Access палитры компонентов, перетащите из нее компонент TDataSource и установите рядом с компонентом TClientDataSet (они будут работать в связке). Подключите свойство DataSet компонента TDataSource к компоненту TClientDataSet. Перейдите на вкладку Data Controls палитры компонентов и перетащите из нее на поле экранной формы компонент TDBGrid. Свяжите его через свойство DataSource с компонентом TDataSource.

Сейчас, на стадии разработки можно просмотреть данные в элементе управления TDBGrid — для этого нужно установить значение **true** в свойстве Active компонента TClientDataSet. Результат показан на рис. 19.18.

Использование вложенных таблиц

Думаю, что самые внимательные читатели, рассматривая рис. 19.18, уже обнаружили, что в экранной форме представлены данные только из таблицы TableCustomers. Если на своем компьютере вы воспользуетесь горизонтальной полосой прокрутки и перейдете к крайней правой колонке таблицы, то увидите, что она озаглавлена TableOrders. Компонент TDBGrid не может отобразить данные в этой колонке и выводит в ней только (DATASET). В действительности эта последняя колонка представляет компонент TDataSetField.

В работающем клиентском приложении MidasClient после щелчка на какой-нибудь ячейке в колонке TableOrders в ней появится многоточие, а после повторного щелчка (или после двойного щелчка на DATASET) на экране откроется еще одно окно, в котором будут выведены записи из подчиненной таблицы, связанные заданным отношением с текущей записью главной таблицы (рис. 19.19).

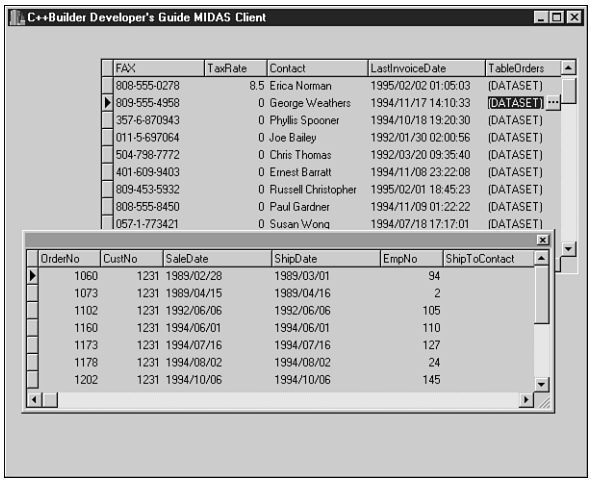


Рис. 19.19. Отображение данных из главной и подчиненной таблиц в окне работающего клиентского приложения MIDAS

Выглядит это прекрасно, но задумайтесь: будет ли удобно конечному пользователю работать с таким интерфейсом? Не лучше ли выводить данные из подчиненной таблицы в отдельный компонент TDBGrid, который будет размещаться под основным. Конечно, о вкусах не спорят, но мне второй вариант кажется более привлекательным. Во всяком случае с помощью C++Builder его несложно реализовать.

Закройте клиентское приложение, если вы запускали его на выполнение, и вновь вернемся в среду разработки C++Builder. Перетащите на поле экранной формы приложения еще один компонент TClientDataSet (по умолчанию ему будет присвоено имя ClientDataSet2). На сей раз обратите внимание на свойство DataSetField нового элемента ClientDataSet2. Требуется каким-то образом подключить к этому свойству поле (колонку) TableOrders, имеющее тип TDataSetField. Для того чтобы использовать вложенный набор данных (записи подчиненной таблицы), нужно создать сохраняемое поле DataSet для этих вложенных данных. Хотя это звучит очень замысловато, создание такого поля не представляет сложности. Дважды щелкнув на первом компоненте TClientDataSet (элементе ClientDataSet1), откройте окно редактора полей. Щелкните на поле редактора правой кнопкой мыши и выберите в контекстном меню команду Add All Fields. В результате будут созданы сохраняемые поля для всех полей главной таблицы, в том числе и поле DataSetField для вложенной подчиненной таблицы. Если дважды щелкнуть на элементе ClientDataSet2, то будет выведено сообщение об ошибке ClientDataSet2: Missing data provider or data packet (ClientDataSet2: отсутствует провайдер данных или пакет данных). Это сообщение появится по той простой причине, что элемент ClientDataSet2 еще не подключен к провайдеру или DataSetField.

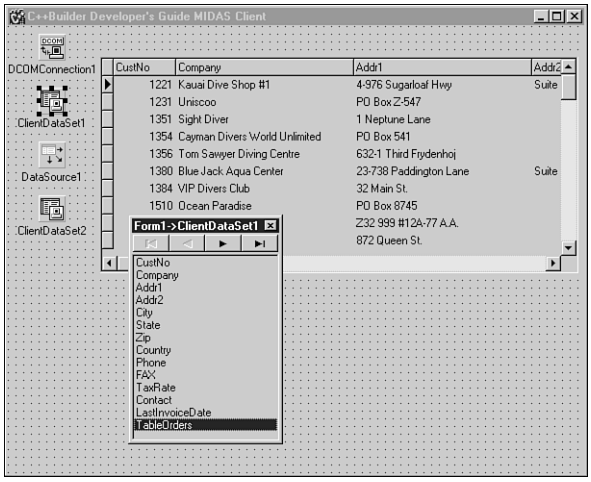


Рис. 19.20. Поля таблицы TableCustomer в окне редактора полей

Вот теперь можно открыть список в поле значения свойства DataSetField второго компонента TClientDataSet (элемента управления ClientDataSet2) — вы увидите в нем элемент ClientDataSet1.TableOrders. Выберите этот элемент в качестве значения свойства DataSetField этого компонента. Учтите, что элемент ClientDataSet не связан напрямую с удаленным сервером, а только косвенно, через первый компонент TClientDataSet.

Теперь перетащите на поле экранной формы компонент TDataSource из вкладки Data Access палитры и второй компонент TDBGrid из вкладки Data Controls. Подключите второй компонент TDataSource (имя этого элемента управления — DataSource2) ко второму компоненту TClientDataSet (элементу управления ClientDataSet2), а второй компонент TDBGrid (элемент управления DBGrid) — к элементу DataSource2 (второму компоненту TDataSource). Теперь на стадии разработки вы увидите в окне приложения данные из подчиненной таблицы.

Описанный выше вариант организации элементов управления в экранной форме клиентского приложения позволяет как отображать данные из двух таблиц, связанных отношением

"главный–подчиненный", так и обновлять их. Конечно, в некоторых случаях можно отдать предпочтение варианту с всплывающим окном, но чаще предложенное выше решение оказывается более приемлемым для конечного пользователя.

Проблема с обновлением данных в обеих таблицах решается таким образом, что нужно вызывать только метод `ApplyUpdates()` первого компонента `TClientDataSet` — единственного, связанного напрямую с удаленным сервером. При этом автоматически произойдет обновление и подчиненной таблицы.

Загрузка локальной сети при использовании MIDAS

Если при работе с приложением `SimpleMidasServer` никаких неприятных эффектов, связанных с загрузкой сети распределенным приложением, не возникает, то при работе со связанными таблицами они уже могут проявляться.

Когда компонент `TClientDataSet` активизируется (его свойство `Active` получает значение `true`), формируется запрос к компоненту `TDataSetProvider` RDM-модуля на пересылку данных по сети. Объем передаваемой информации зависит от длины записи набора данных и, естественно, от количества записей. Последний параметр определяется значением свойства `PacketRecords` компонента `TClientDataSet`. По умолчанию этому свойству присваивается значение `-1`, что означает "передавать все доступные записи".

Когда мы работаем в описанных выше приложениях с базой данных `BCDEMOS`, в которой таблица `customers.db` содержит только 55 записей, а таблица `orders.db` только 205 записей, передача информации практически не сказывается на производительности сети. Но реальные таблицы содержат значительно больший объем информации — тысячи и десятки тысяч записей. При длине записи в несколько сотен байт придется при подключении клиента к серверу MIDAS передавать по сети сотни килобайт данных. Думаю даже на таком простом примере вы убедились, что при подобной организации могут возникнуть серьезные задержки в работе сети.

Использование свойства `PacketRecords`

Существует несколько способов справиться с проблемой перегрузки сети БД-приложениями MIDAS. Самый очевидный — настроить значение свойства `PacketRecords` компонента `TClientDataSet` в зависимости от того, сколько записей нужно выводить одновременно на экран (обычно на экране умещается порядка 20 записей). Если присвоить именно такое значение свойству `PacketRecords`, то при подключении клиента к серверу последний экспортирует только первые 20 записей заданного набора данных. Как только пользователь на стороне клиента начнет перемещаться по строкам элемента `TDBGrid` и дойдет до 21-й строки, компонент `TClientDataSet` сформирует новый запрос к компоненту `TDataSetProvider` RDM-модуля и получит следующие 20 записей. Теперь в элемент `TDBGrid` можно будет выводить уже 40 записей, хотя все они на экране могут быть и не видны. Этот процесс продолжается до тех пор, пока все записи из RDM-модуля не окажутся в компоненте `TClientDataSet` клиентского приложения.

На заметку Изменять значение свойства `PacketRecords` второго компонента `TClientDataSet`, который имеет отношение к подчиненной таблице, нет нужды. Вложенный набор данных будет передан клиенту в качестве `DataSetField` в составе записи главной таблицы.

После такой модификации обратите внимание на ползунок вертикальной полосы прокрутки сетки `TDBGrid` главной таблицы в экранной форме клиентского приложения — размер ползунка увеличился. Это произошло по той причине, что при подключении клиента к серверу было передано только 20 записей и значительная часть их уместилась на экране (сравните рис. 19.21 и 19.19).

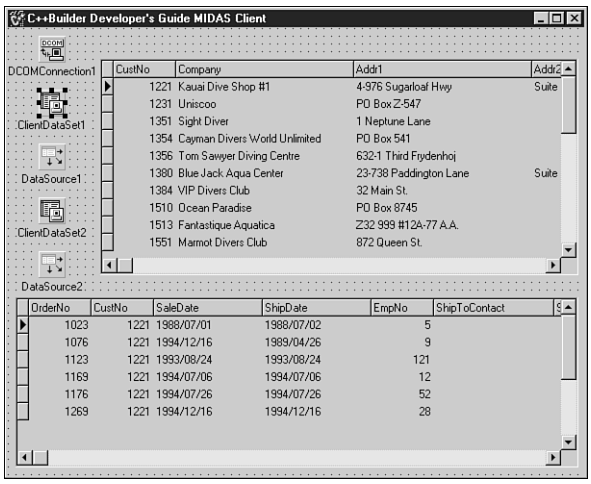


Рис. 19.21. В клиентское приложение `MidasClient` загружены только первые 20 записей главной таблицы

На первый взгляд, мы получили вполне приемлемое решение, позволяющее избежать перегруженности сети. Но в этом решении имеется довольно существенный изъян. Представьте себе следующую ситуацию. Запущено на выполнение клиентское приложение `MidasClient`. У пользователя перед глазами первые 20 записей таблицы, а он желает посмотреть данные в самой последней записи. По привычке пользователь щелкает кнопкой мыши на поле таблицы и нажимает <Ctrl+End>. После этого ему придется немного подождать, если в таблице 55 записей, и подождать значительно дольше, если в таблице 55 000 записей. Дело в том, что компонент `TClientDataSet` будет запрашивать данные отдельными порциями по 20 записей, а такая передача "по чайной ложке" занимает больше времени, чем передача сразу всех записей из таблицы. Причем, если помните, пользователь-то ведь "просил" только *последнюю* запись, а не все записи таблицы.

Оптимизация сервера

Помимо оптимизации на стороне клиента с помощью свойства `PacketRecords`, можно попробовать изменить что-либо и на стороне сервера. Вы помните, что объем пересылаемой информации определяется как количеством записей, так и объемом данных в каждой пересылаемой записи. На стороне клиента с помощью свойства `PacketRecords` выполняется управление количеством записей, а на стороне сервера можно организовать управление размером передаваемой записи. Очень часто оказывается, что совсем необязательно передавать все поля записи, поскольку на экране пользователь одновременно может наблюдать только часть из них. В наших примерах компонент `TDataSetProvider` экспортировал все поля записей обеих таблиц — и `TableCustomer`, и `TableOrders`. Очевидно, что нужно включить в состав серверного приложения программу, которая принимала бы решение, сколько и какие именно поля таблиц имеет смысл передавать клиенту.

Совместными усилиями клиента и сервера MIDAS можно добиться существенного снижения загрузки локальной сети при передаче данных.

Оставшаяся часть этой главы будет посвящена усовершенствованиям, внесенным в версию MIDAS 3 и отличающим эту версию от предыдущих.

Новинки MIDAS 3

Первая версия MIDAS была включена в состав программного продукта C++Builder 3, вместе с версией 4 C++Builder распространялась и версия 2 средств поддержки технологии MIDAS. Новая, третья версия MIDAS включена в комплект поставки программного продукта C++Builder 5 Enterprise, и эта новая версия предлагает разработчикам распределенных БД-приложений множество усовершенствований по сравнению с предыдущими. Материал этого раздела адресован в первую очередь тем читателям, которые имеют определенный опыт работы с предыдущими версиями MIDAS. В нем мы расскажем о новых функциональных возможностях MIDAS 3 в составе C++Builder 5 Enterprise.

Компонент `TDataSetProvider`

В состав программного продукта C++Builder 4 (и MIDAS 2) входили два варианта компонентов провайдера услуг сервера MIDAS, слегка отличавшихся по своим возможностям: `TProvider` и `TDataSetProvider`. Кроме того, первый набор данных, которые желательно было экспортировать из RDM-модуля (речь все еще идет о C++Builder 4), мог использовать интерфейс по умолчанию `IProvider` (хотя это и не рекомендовалось). В MIDAS 3 дело обстоит по-другому.

Во-первых, как вы уже видели в примерах, описанных в предыдущих разделах, в MIDAS 3 имеется только один компонент провайдера — `TDataSetProvider`. Во-вторых, теперь разработчик должен в явном виде использовать компонент `TDataSetProvider` для каждого набора данных в компонентах `TTable`, `TQuery`, `TStoredProc` и им подобных, который предполагается экспортировать из RDM-модуля. В-третьих, в MIDAS 3 несколько изменена технология работы с компонентом. Работая с MIDAS 2 в среде C++Builder 4, разработчик должен был перетащить компонент `TDataSetProvider` (или `TProvider`) на поле модуля данных и подключить его к набору данных, а затем щелкнуть на поле компонента правой кнопкой мыши и выбрать в контекстном меню команду Export From Remote Data Module (Экспорт из RDM-модуля). Последняя операция в C++Builder 5 исключена — после того как компонент `TDataSetProvider` будет включен в состав RDM-модуля и подключен к компоненту набора данных, управление экспортом выполняется через свойство `Exported`, которому может быть присвоено значение `true` или `false`. В первом случае будут экспортироваться данные из RDM-модуля, а во втором — экспортирование будет заблокировано. Установка значения свойства `Exported` возможна как на этапе разработки, так и программно в процессе работы приложения.

Четвертое изменение в `TDataSetProvider` касается интерфейсов. При работе в среде C++Builder 4 каждый экспортирующий провайдер соответствовал какому-либо свойству интерфейса — его нельзя было удалить. Более того, это свойство имело тип `IProvider` и использовалось клиентским приложением в процессе взаимодействия с сервером. В C++Builder 5 прикладной сервер сам по себе является единственным интерфейсом `IAppServer`, который организует взаимодействие с клиентами. В результате сервер MIDAS 3 располагает возможностью программно отключать провайдер в процессе выполнения программы, манипулируя значением его свойства `Exported`.

Интерфейсы `IProvider` и `IAppServer`

Серверы MIDAS, созданные в среде C++Builder 4, не выполняли самостоятельно регистрацию в системном реестре, вызывая функцию `UpdateRegistry()`. Такой сервер экспортировал интерфейс `IProvider`, а не новый интерфейс `IAppServer`. В интерфейс `IAppServer` внесено одно

очень существенное усовершенствование, по сравнению с интерфейсом IProvider: теперь сервер стал объектом, не имеющим состояния (об этом подробнее в следующем разделе). При использовании MIDAS 2 и интерфейса IProvider сервер изменял собственное состояние в соответствии с запросами клиентского приложения (фиксировал, какие записи переданы клиентскому приложению, какая следующая запись стоит в очереди на передачу и т.д.). В результате сервер MIDAS 2 нельзя было использовать в сочетании с технологиями, ориентированными на работу с объектами, не сохраняющими состояния (имеются в виду технологии MTS (Microsoft Transaction Server), CORBA, HTTP и им подобные).

Одним из побочных следствий применения в MIDAS 3 интерфейса IAppServer является несовместимость приложение MIDAS 2 с приложениями MIDAS 3.

Брокеры данных, не сохраняющие состояния

Как уже упоминалось, приложения MIDAS 3 способны работать с RDM-модулями, не имеющими внутреннего состояния. Это одно из наиболее существенных изменений в MIDAS 3. В результате отпадает необходимость в специальном интерфейсе для RDM-модуля на базе интерфейса IProvider. Кроме того, каждый клиент теперь должен самостоятельно следить за состоянием набора данных и передавать соответствующую информацию серверу при каждом запросе. В сопроводительной документации на MIDAS 3 утверждается, что, хотя каждое обращение клиента к серверу связано с передачей бо́льшего объема информации (нужно передавать информацию о состоянии набора данных), количество вызовов существенно сокращается, а потому фактическая загрузка сети уменьшается.

Чтобы более четко представить характер внесенных в новой версии изменений, я сравнил перечни событий версий компонента TClientDataSet в C++Builder 4 (MIDAS 2) и в C++Builder 5 (MIDAS 3). Набор события компонента TClientDataSet в C++Builder 5 включает AfterApplyUpdates, AfterExecute, AfterGetParams, AfterGetRecords, AfterPost, AfterRefresh, AfterRowRequest, AfterScroll, BeforeApplyUpdates, BeforeExecute, BeforeGetParams, BeforeGetRecords, BeforePost, BeforeRefresh, BeforeRowRequest и BeforeScroll. Таким образом, для передачи от клиента к серверу информации о состоянии необходимо использовать восемь наборов методов.

Аналогичный список отличий можно составить и для сервера. Я сравнил список событий версий компонентов TDataSetProvider в C++Builder 4 и C++Builder 5. В новой версии TDataSetProvider имеются обработчики событий AfterApplyUpdates, AfterExecute, AfterGetParams, AfterGetRecords, AfterRowRequest, BeforeApplyUpdates, BeforeExecute, BeforeGetParams, BeforeGetRecords и BeforeRowRequest. Кроме того, имеется еще и обработчик события OnGetTableName.

Обрабатывая события Before... компонента TDataSetProvider, можно получить от клиента информацию о состоянии (она передается соответствующими обработчиками событий Before... компонента TClientDataSet на стороне клиента). При обработке на сервере событий After... компонента TDataSetProvider можно отправить информацию о состоянии клиенту (эта информация извлекается при обработке соответствующий событий After... компонентом TClientDataSet на стороне клиента).

В качестве примера рассмотрим, как будут обмениваться информацией компоненты TClientDataSet и TDataSetProvider при передаче очередной порции записей из набора данных. Во-первых, установите в свойстве FetchOnDemand объекта ClientDataSet1 (это компонент TClientDataSet главной таблицы) значение **false**. Если этого не сделать, то позже появится сообщение об ошибке заполнения стека.

Если свойству FetchOnDemand присвоено значение **false**, то автоматический механизм пересылки пакетов записей от сервера MIDAS клиенту блокируется, и пересылку необходимо организовать самостоятельно. Когда может понадобиться такая нестандартная процедура пересылки записей? Предположим, что в приложении перед тем как отсылать записи клиенту, необходимо проверить, правильно ли позиционирован набор данных на стороне сервера. Методика, которая будет описана ниже, очень помогает при работе с технологиями MTS, CORBA или HTTP, ориентированными на объекты, не сохраняющие состояния.

Итак, для получения очередной порции записей от компонента TDataSetProvider сервера MIDAS 3 компонент TClientDataSet клиентского приложения MidasClient должен обработать событие BeforeGetRecords. (Свойству FetchOnDemand компонента TClientDataSet ранее присвоено значение **false**.) В процессе обработки нужно специфицировать определенную запись в наборе или позицию. Для обмена информацией используется аргумент OwnerData обработчика события. Программный код обработчика представлен в листинге 19.4.

Листинг 19.4. Обработчик события `OnBeforeGetRecords()` компонента `ClientDataSet`

```
void __fastcall TForm1::ClientDataSet1BeforeGetRecords(
      TObject *Sender,
      OleVariant &OwnerData)
{
  TClientDataSet* Master = (TClientDataSet*) Sender;
  if (Master->Active)
  {
    void* Current = Master->GetBookmark();
    try
    {
      Master->Last();
      OwnerData = Master->FieldByName("CustNo")->AsString;
      Master->GotoBookmark(Current);
    }
    __finally
    {
      Master->FreeBookmark(Current);
    }
  }
}
```

Если свойство FetchOnDemand компонента TClientDataSet имеет значение **true**, то при выполнении оператора Master->Last() возникнет ошибка переполнения стека. В таком случае переход к последней записи заставит компонент TClientDataSet затребовать все записи, а это приведет к новому возбуждению события OnBeforeGetRecords и вложенному вызову соответствующего обработчика, т.е. цепочка событий замкнется, что и приведет в конце концов к переполнению стека.

Обработчик события BeforeGetRecords вызывается перед тем как компонент TDataSetProvider RDM-модуля сервера MidasServer отошлет затребованные записи данных, причем обработчику будет передан аргумент OwnerData, полученный от компонента ClientDataSet на стороне клиента (листинг 19.5).

Листинг 19.5. Обработчик события `OnBeforeGetRecords()` компонента `DataSetProvider`

```
void __fastcall TCustomerOrders::DataSetProvider1BeforeGetRecords(
      TObject *Sender, OleVariant &OwnerData)
{
  TVariant Variant = OwnerData;
  if (!VarIsEmpty(Variant))
  {
    TLocateOptions LocateOptions;
    TDataSet* DataSet = ((TDataSetProvider*) Sender)->DataSet;
    if (DataSet->Locate("CustNo", Variant, LocateOptions))
      DataSet->Next();
  }
}
```

После того как оба обработчика событий `BeforeGetRecords` (и на стороне клиента, и на стороне сервера) сделают свое дело, наступит время передавать данные от RDM-модуля сервера `MidasServer` клиентскому приложению `MidasClient`.

После завершения передачи данных компонент `TDataSetProvider` на стороне сервера может послать информацию о состоянии обратно клиенту (его компоненту `TClientDataSet`). В частности, можно передать общее количество записей, которое имеется на сервере. Передача этой информации выполняется обработчиком события `AfterGetRecords` компонента `TDataSetProvider` (листинг 19.6).

Листинг 19.6. Обработчик события `OnAfterGetRecords()` компонента `DataSetProvider`

```
void __fastcall TCustomerOrders::DataSetProvider1AfterGetRecords(
      TObject *Sender, OleVariant &OwnerData)
{
  TDataSet* DataSet = ((TDataSetProvider*) Sender)->DataSet;
  if (DataSet->Active)
    OwnerData = IntToStr(DataSet->RecordCount);
  else
    OwnerData = AnsiString("n/a");
}
```

Обратите внимание на то, что значение типа `AnsiString` передается через аргумент `OwnerData`, который имеет тип `OleVariant`. При передаче значения типа `AnsiString` у меня никогда проблем не возникало, а вот использование конструкции `DataSet->RecordCount` в качестве аргумента `OwnerData` приводило к ошибке компиляции.

Значение `OwnerData`, переданное обработчиком события `OnAfterGetRecords` компонента `TDataSetProvider`, извлекается на стороне клиента обработчиком события `OnAfterGetRecords` компонента `TClientDataSet`. Как сохранить полученное значение — другой разговор, а в данном примере я просто вывожу его пользователю в окне сообщения (листинг 19.7).

Листинг 19.7. Обработчик события `OnAfterGetRecords()` компонента `ClientDataSet`

```
void __fastcall TForm1::ClientDataSet1AfterGetRecords(
      TObject *Sender,
```

```
      OleVariant &OwnerData)
{
  ShowMessage("Number of records at server: " +
                            WideString(OwnerData));
}
```

Новые версии приложений `MidasServer` и `MidasClient`, в которые добавлены обработчики событий `OnBeforeGetRecords()` и `OnAfterGetRecords()` компонентов `TClientDataSet` (на стороне клиента) и `TDataSetProvider` (на стороне сервера), теперь можно сохранить и скомпилировать. Но перед сохранением не забудьте установить значение **`false`** в свойстве `FetchOnDemand` компонента `TClientDataSet`, что предотвратит переполнение стека.

Тестирование новой версии БД-приложения выполняется следующим образом. Запустите на выполнение клиентское приложение `MidasClient`, которое самостоятельно загрузит при подключении серверное приложение `MidasServer`. Как только будет установлено соединение между клиентом и сервером, компонент `TClientDataSet` сформирует первый запрос (при этом вызывается метод `GetNextPacket()`). В результате будет запущен обработчик события `OnBeforeGetRecords()` компонента `TClientDataSet`, который не передаст никакой информации, поскольку в это время свойство `Active` имеет значение **`false`**. Далее, хотя на стороне сервера и будет вызван обработчик `OnBeforeGetRecords()` компонента `TDataSetProvider`, он получит "пустой" аргумент `OwnerData`, и никакие операции выполнены не будут. Затем компонент `TDataSetProvider` отошлет первые 20 записей (именно такая длина пакета записей задана в свойстве `PacketRecords`) компоненту `TClientDataSet` на стороне клиента. При этом будет запущен обработчик события `OnAfterGetRecords()` компонента `TDataSetProvider`, который передаст информацию о длине набора записей на сервере (в данном случае о длине таблицы `TableCustomer`, которая насчитывает 55 записей). Значение **`55`** передается через аргумент `OwnerData` и извлекается обработчиком события `OnAfterGetRecords()` компонента `TClientDataSet` на стороне клиента. В результате на экране конечного пользователя появится окно сообщения с текстом `Number of records at server: 55` (Количество записей на сервере: 55) (рис. 19.22).

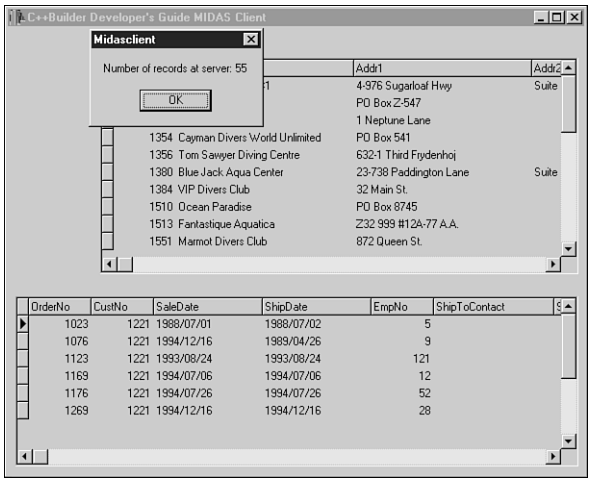


Рис. 19.22. Окно сообщения, сформированное обработчиком события `OnAfterGetRecords()` компонента `TClientDataSet`

После этого в сетке окна клиентского приложения `MidasClient` пользователь сможет просмотреть первые 20 записей главной таблицы и соответствующие им записи из подчиненной таблицы. Если передвинуть курсор в поле главной таблицы на 21-ю запись, никакие новые

данные в таблице на экране не появятся. Точно так же ничего не произойдет и при нажатии клавиши <Ctrl+End>. Причина в том, что свойство FetchOnDemand имеет значение **false**. Поэтому программист должен самостоятельно организовать обращение к методу GetNextPacket(). Проще всего это сделать, добавив в экранную форму клиентского приложения еще одну кнопку — компонент Tbutton. Присвойте ей имя **ButtonFetch**, а свойству Caption компонента — значение **Fetch**, как показано на рис. 19.23. Программный код обработчика события OnClick этой кнопки представлен ниже:

```
void __fastcall TForm1::ButtonFetchClick(TObject *Sender)
{
  ClientDataSet1->GetNextPacket();
}
```

После щелчка на кнопке Fetch в клиентское приложение будут переданы следующие 20 записей, а после следующего щелчка — остальные 15 записей. На этот раз объект DataSet в MidasServer уже не будет активным, а потому значение RecordCount не будет сформировано (вот почему в обработчик события OnAfterGetRecords компонента TDataSetProvider вставлен блок else).

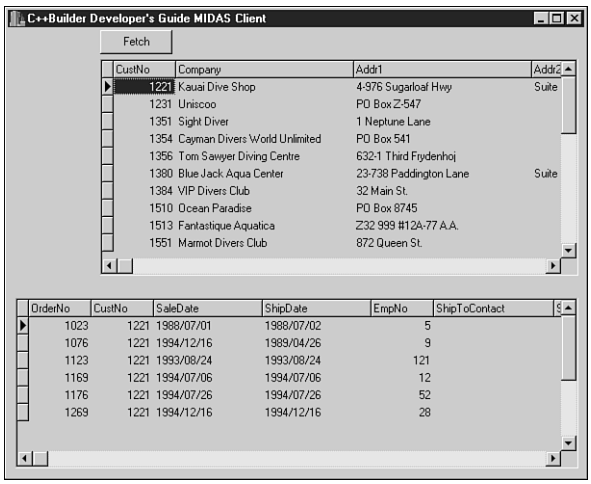


Рис. 19.23. Окно клиентского приложения с кнопкой загрузки новой порции записей

Какой же вывод нужно сделать из проведенных экспериментов с БД-приложением MIDAS 3? Как вы убедились, сервер не располагает собственной информацией о текущем состоянии набора данных. Такая информация может быть получена им только от клиента, причем сервер и клиент обмениваются информацией через аргумент OwnerData обработчиков событий Before... и After.... Еще раз обращаю ваше внимание на то, что этот аргумент имеет тип OleVariant. Следовательно, через него можно передавать данные любого типа, но на принимающей стороне, конечно, нужно знать, как интерпретировать полученные данные. Поэтому я чаще всего передаю через него данные типа AnsiString.

Помимо событий групп Before... и After..., компонент TClientDataSet может обрабатывать еще и два других событий — Post и Scroll, которые не имеют прямых аналогов на стороне сервера (у его компонента TDataSetProvider). В качестве аргументов обработчики этих событий принимают только набор данных DataSet.

И еще одно замечание. Установка значения **true** в свойстве FetchOnDemand компонента TClientDataSet обеспечивает автоматическую передачу информации о состоянии от клиента к серверу. Но иногда нужно определенным образом управлять этой информацией, и в этом случае на помощь приходит продемонстрированная выше методика. Особенно существенно держать эту информацию под контролем, работая с протоколами MTS, CORBA или HTTP.

Использование технологии InternetExpress

До сих пор мы рассматривали единственный тип клиентского приложения, в котором для загрузки и хранения пакета записей используется компонент TClientDataSet. Этот компонент идеально подходит для приложений Win32 или приложений типа ActiveForm (такие приложения рассматриваются в главе 22).

Но кроме TClientDataSet, в состав C++Builder 5 входит и другой компонент работы с пакетами записей — TXMLBroker (он включен только в редакцию C++Builder 5 Enterprise). Этот компонент применяется в том случае, если пакет записей имеет формат XML. Как и TClientDataSet, компонент XMLBroker подключается к компоненту соединения (в эту группу входят TDCOMConnection, TWebConnection и TSocketConnection) и может использовать свойство ProviderName для настройки на извлечение данных из определенного компонента провайдера типа TDataSetProvider. Единственным заметным отличием между TXMLBroker и TClientDataSet является то, что для получения/отправки пакета данных первый использует XML-формат, а второй — двоичный.

Где и в какой ситуации следует использовать компонент TXMLBroker? Этот компонент представляет собой один из элементов технологии, получившей наименование *InternetExpress*, которую можно рассматривать как симбиоз технологии WebBroker (глава 13) и MIDAS. В разделе *InternetExpress* главы 13 мы уже рассматривали приложение, которое работает и как клиент MIDAS, и как Web-сервер (Web-модуль). Я буду часто ссылаться на материал этой главы, поэтому рекомендую освежить его в памяти.

Свойство MaxRecords компонента XMLBroker

В этом разделе внимание будет сосредоточено на тех особенностях клиентского приложения ***InternetExpress***, которые имеют отношение к MIDAS. Прежде всего вновь обратимся к проблеме загрузки сети при обмене информацией между сервером и клиентами MIDAS. Используя компонент TClientDataSet, в частности его свойство PacketRecords, можно задать количество записей, передаваемых в составе отдельного пакета от сервера (его компонента TDataSetProvider) клиенту (его компоненту TClientDataSet). Как было сказано в главе 13, компонент XMLBroker таким свойством не располагает. Но эту функцию может выполнить свойство MaxRecords, которое ограничивает количество записей, пересылаемых от TDataSetProvider к компоненту TXMLBroker. К сожалению, от TDataSetProvider к TXMLBroker можно переслать только один пакет записей в XML-формате, и не существует способа, который позволил бы передать другой пакет. Даже если включить самостоятельно в программу вызов метода GetNextPacket() (как это делалось ранее в одном из обработчиков события компонента TClientDataSet), в свойстве DataSet компонента TDataSetProvider будет восстанавливаться указатель на первую запись ранее переданного пакета (эту операцию выполняет обработчик события OnBeforeGetRecords() перед началом передачи данных от TDataSetProvider к TXMLBroker). Это происходит из-за того, что в методе GetXMLRecords() компонента TDataSetProvider предусмотрена установка флага grReset. Этот флаг заставляет восстанавливать положение курсора после каждой передачи. Единственное, что я смог придумать, чтобы избавиться от этого восстановления, — создать класс, производный от TDataSetProvider (назовем его TBDataSetProvider), в котором метод GetXMLRecords() не будет устанавливать флаг grReset.

Но другая методика ограничения загрузки сети, предусматривающая минимизацию количества полей в записи, работает и в сочетании с TXMLBroker. При работе с компонентами InternetExpress это имеет довольно существенное значение, поскольку объем записи в XML-формате увеличивается почти вдвое по сравнению с двоичным форматом. Например, объединение

CustomerOrders двух таблиц связью "главный–подчиненный" занимает порядка 72 Кбайт при том, что в главной таблице содержится 55 записей, а в подчиненной — около 200 записей. Причем в этот объем не входят дополнительные файлы JavaScript, которые необходимы для подключения XML-данных к элементам управления HTML, — а это порядка 60 Кбайт.

Свойство MaxErrors компонента XMLBroker

Свойство MaxErrors компонента TXMLBroker играет ту же роль, что и аргумент MaxErrors метода ApplyUpdates() компонента TClientDataSet. Его значение задает максимальное количество ошибок, допустимое при обновлении, прежде чем процесс обновления будет прерван. Кнопка Apply Updates при использовании ***InternetExpress*** располагается в Web-броузере. После щелчка на этой кнопке приложение WebBroker заставляет компонент TXMLBroker передать обновленные данные в составе XML-пакета обратно компоненту провайдера TDataSetProvider — здесь наблюдается полная аналогия с функционированием компонента TClientDataSet. Единственное отличие в том, что на сей раз данные передаются в XML-формате, а не в двоичном, и нельзя выводить на экран диалоговое окно Update Error в Web-броузере. А может быть можно? Оказывается, если подумать немножко, то все-таки можно!

В каталоге CBuilder5/Examples/MIDAS/InternetExpress/Inetxcustom имеется специальный *пакет времени разработки* dclinetxcustom_bcb.bpk, который понадобится вам для того чтобы правильно загрузить проект Webshow.bpr. Этот пакет нужно скомпилировать и установить. В нем имеется специальный компонент, производный от PageProducer, который называется TReconcilePageProducer (его можно отыскать на вкладке InternetExpress палитры компонентов, после того как пакет dclinetxcustom_bcb установлен). Учтите, что в список путей Include path на вкладке Directories/Conditionals диалогового окна Project Options нужно добавить еще один элемент — $(BCB)\examples\midas\internetexpress\inetxcustom. Это позволит без всяких проблем скомпилировать приложение, в котором используется компонент TReconcilePageProducer.

После этого загрузите в среду C++Builder проект ShowWeb, который был создан при выполнении упражнений главы 13, и перетащите на поле Web-модуля компонент TReconcilePageProducer (этот компонент появится на вкладке InternetExpress палитры компонентов после установки пакета dclinetxcustom_bcb.bpk). Можете поэкспериментировать со свойствами нового компонента PageProducer, но главное, что предстоит сделать, — присвоить ссылку на этот компонент для свойства ReconcileProducer компонента TXMLBroker.

Теперь, как только возникнет какой-либо конфликт при обновлении записей в сервере MidasServer и сервер возвратит компоненту TXMLBroker сообщение об ошибке, "за дело возьмется" компонент ReconcilePageProducer и динамически сформирует HTML-текст, который даст пользователю информацию о сути конфликта.

Компонент WebConnection

Компонент TDCOMConnection поддерживает протокол DCOM в БД-приложении на базе MIDAS, но этот протокол не является единственно доступным. MIDAS также поддерживает и два других распространенных протокола: TCP/IP (сокеты) и HTTP. Последний поддерживается компонентом TWebConnection, и его имеет смысл использовать с proxy-серверами и брандмауэрами. Использовать в такой ситуации компонент TDCOMConnection довольно сложно (хотя и возможно).

Компонент TWebConnection находится на вкладке Midas палитры компонентов. Особую ценность этому компоненту придает его способность работать в составе сервера MIDAS 3, который не сохраняет состояния, в несохраняющей состояние среде протокола HTTP.

Для выполнения последующих упражнений вам нужно скопировать файлы созданного ранее проекта MidasClient в какой-нибудь другой каталог, где можно поэкспериментировать с исполь-

зованием компонента TWebConnection в этом проекте. Замечание о переносе относится только к клиентскому приложению — в сервере MidasServer мы вообще ничего менять не будем.

В скопированном в новый каталог проекте MidasClient замените существующий компонента TDCOMConnection новым — типа TWebConnection. Теперь нужно организовать его соединение с сервером MidasServer — тем самым, который был создан в предыдущих разделах этой главы. Компонент TWebConnection организует соединение на основе протокола HTTP. Но перед началом работы проверьте, установлен ли DLL-модуль WININET.DLL на том компьютере, где находится клиентское приложение (этот модуль должен быть установлен, если у вас работает ***Internet Explorer*** версии 3 или более поздней). На компьютере сервера должен быть установлен ***Internet Information Server*** версии 4 или более поздней или ***Netscape Enterprise*** версии 3.6 или более поздней. Последнее, что нужно сделать, — установить в каталоги cgi-bin или scripts на Web-сервере, который будет использоваться для соединения с компонентом TWebConnection, файл HTTPSRVR.DLL. Это — специальный DLL-модуль ISAPI, разработанный в Inprise/Borland, и его можно найти в каталоге CBUILDER5\BIN.

Модуль HTTPSRVR выполняет запуск сервера MidasServer на Web-сервере и маршалинг всех запросов клиентов к интерфейсу прикладного сервера, отправляя обратно пакеты записей.

Более подробная информация о Web-серверах, DLL-модулях ISAPI и методике программирования Web-серверов содержится в главе 13.

Свойство URL компонента WebConnection должно содержать адрес http://localhost/cgi-bin/httpsrvr.dll (на моем компьютере этот файл находится в каталоге scripts, но я мог бы также использовать и адрес http://192.168.92.201/cgi-bin/httpsrvr.dll). Далее щелкните на свойстве ServerName, откройте список доступных серверов MIDAS и выберите из него сервер MidasServer.CustomerOrders. Убедиться в том, что соединение организовано, можно прямо сейчас — дважды щелкните на свойстве Connected компонента TWebConnection. Если значение этого свойства изменится на **true**, значит все в порядке.

На заметку

Учтите, что в действительности вы не увидите признаков работающего сервера MIDAS. Это происходит потому, что HTTPSRVR (этот модуль запускается Web-сервером) активизирован другим пользователем (пользователем Internet, назначенным по умолчанию). Ранее при использовании DCOM в аналогичной ситуации на экране появлялось окно прикладного сервера.

Убедиться, что сервер работает, можно также с помощью ***Task Manager***. В окне Task Manager вы увидите MidasServer в списке выполняемых в текущий момент приложений, хотя на экране окно сервера отсутствует.

В остальном компонент TWebConnection работает точно так же, как и компонент TDCOMConnection, хотя есть еще один нюанс — безопасность. Компонент TWebConnection позволяет использовать все возможности протокола безопасности SSL и работать с приложением MidasServer, которое защищено брандмауэром. В структуре компонента TWebConnection имеются и другие полезные свойства — Proxy, ProxyByPass, UserName и Password. Свойства UserName и Password компонента TWebConnection можно использовать при работе с Web-серверами, которые требуют авторизации и аутентификации.

Пул объектов

Соединение с Web позволяет использовать и пул объектов сервера для обслуживания запросов клиентов. При этом MidasServer не использует “вхолостую” системные ресурсы для RDM-модуля и соединения с базой данных. Эти ресурсы затребуются только тогда, когда в них возникает необходимость.

Работа с пулом объектов позволяет установить максимальное количество экземпляров RDM-модулей внутри приложения сервера MIDAS. Как только поступает запрос от клиентского приложения, сервер MIDAS проверяет, есть ли свободный RDM-модуль в пуле. Если таковой найти не удается, формируется новый экземпляр RDM-модуля (но количество созданных экземпляров не может превышать заданное максимальное значение) или генерируется исключительная ситуация с сообщением `Server too busy` (Сервер перегружен). В свою очередь, RDM-модуль обслуживает запрос клиента и затем переходит в состояние ожидания. Если после этого в течение определенного времени новый запрос не поступит, экземпляр RDM-модуля автоматически уничтожается, и занятые им ресурсы высвобождаются. Именно последней операцией и занимается программа обслуживания пула объектов.

В более ранних версиях MIDAS эта возможность отсутствовала, а теперь с помощью одного экземпляра RDM-модуля можно обслуживать запросы от нескольких клиентских приложений. Все это стало возможно потому, что сервер не должен хранить никакой информации о состоянии набора данных — она обрабатывается и хранится на стороне клиента.

Возникает вопрос: "Как использовать пул объектов при соединении на основе протокола HTTP?" Для ответа на него придется заглянуть внутрь функции `UpdateRegistry()`, текст которой вы можете отыскать в файле заголовка RDM-модуля. В этой функции объект `regObj` используется для настройки режима работы прикладного сервера. Для работы с пулом объектов необходимо установить еще три свойства этого объекта.

Во-первых, свойство `regObj.MaxObjects` задает максимальное количество экземпляров объекта RDM-модуля. Когда сервер MIDAS получает очередной запрос от какого-либо клиента, а в работе находятся указанное в этом свойстве количество RDM-модулей, генерируется исключительная ситуация с сообщением `Server too busy` (Сервер перегружен).

Во-вторых, свойство `regObj.Timeout` задает интервал времени (в минутах), в течение которого RDM-модуль может ждать очередного запроса со стороны пользователя. По истечении этого интервала экземпляр RDM-модуля автоматически уничтожается программой обслуживания пула объектов сервера MIDAS. Как сказано в сопроводительной документации, сервер MIDAS каждый 6 минут проверяет нет ли "бездельничающего" экземпляра RDM-модуля, который можно было бы безболезненно уничтожить. Если это свойство имеет значение `0`, то экземпляр RDM-модуля уничтожается сразу после выполнения запроса.

Установив значения этих двух свойств, следует разрешить работу с пулом объектов — свойству `regObj.RegisterPooled` присвоить значение `true`.

Есть и еще одно свойство объекта `regObj`, которое имеет косвенное отношение к настройке режима использования пула объектов. Свойство `regObj.Singleton` задает режим "одиночного экземпляра" (singleton) RDM-модуля. Если этому свойству присвоить значение `true`, то сервер будет попросту игнорировать значения свойств `regObj.MaxObjects` и `regObj.Timeout` и создавать только один экземпляр RDM-модуля для обслуживания всех поступающих запросов пользователя. Но, как правило, свойству `regObj.Singleton` присваивается значение `false`.

В листинге 19.8 приведен пример модификации функции `UpdateRegistry()`, в котором задается следующий режим формирования экземпляров RDM-модуля: максимальное количество экземпляров — 10, а время выжидания — 42 мин.

Листинг 19.8. Функция `UpdateRegistry()` с настройкой режима использования пула объектов

```
// Функция регистрирует объект или
// удаляет регистрационную запись объекта.
//
```

```
static HRESULT WINAPI UpdateRegistry(BOOL bRegister)
{
  TRemoteDataModuleRegistrar regObj(GetObjectCLSID(),
                          GetProgID(), GetDescription());
  // Если не предполагается использовать объект для
  // подключения к Winsock или Web, сбросьте эти флаги.
  // Подробная информация о назначении других флагов
  // содержится в файлах atlmod.h и atlvcl.cpp.

  regObj.Singleton = false;
  regObj.MaxObjects = 10;
  regObj.Timeout = 42;
  regObj.RegisterPooled = true;  regObj.EnableWeb = true;
  regObj.EnableSocket = true;
  return regObj.UpdateRegistry(bRegister);
}
```

Конечно, нельзя считать удачным решение "жестко" запрограммировать числовые значения 10 и 42 свойств в этой функции. Во-первых, если потребуется изменить значения этих свойств (если, например, в компьютере сервера увеличился объем оперативной памяти), придется перекомпилировать серверное приложение. Кроме того, часто один и тот же сервер MIDAS устанавливается на нескольких компьютерах, имеющих разную конфигурацию. Короче говоря, я рекомендую использовать внешний файл конфигурации при установке сервера, в котором можно было бы задавать значения параметров для каждой инсталляции. Это позволит оптимально настроить механизм пула объектов.

Использование протокола TCP/IP

Как уже было сказано выше, MIDAS 3 позволяет использовать и третий тип протокола — TCP/IP. Для этого в составе C++Builder 5 имеется специальный компонент `TSocketConnection`. Как и два первых, он находится на вкладке Midas палитры компонентов C++Builder 5 Enterprise.

Если у вас нет доступа к NT-серверу домена в сети, то вы не сможете воспользоваться DCOM, но это не мешает работе с протоколом TCP/IP. Подключение через сокет будет работать даже в том случае, если вообще отсутствует NT-сервер и организовать его значительно легче, чем DCOM-подключение. Но при этом значительно больше усилий придется затратить на настройку системы защиты.

Модифицировать клиентские приложения `SimpleMidasClient` или `MidasClient` с тем, чтобы в них использовался протокол TCP/IP, большого труда не представляет. При этом в сервере MIDAS вообще ничего менять не нужно.

Прежде всего запустите программу `ScktSrvr.exe`, файл которой находится в каталоге `CBuilder5/Bin` на компьютере сервера. Без нее БД-приложение, использующее протокол TCP/IP, работать не будет. Учтите, что программа `ScktSrvr.exe` может выполняться и в статусе обычного приложения, и в статусе службы NT (использующего ключи `-install` и `-uninstall` в командной строке).

Перетащите компонент `TSocketConnection` с вкладки Midas палитры компонентов на поле главной формы клиентского приложения `MidasClient`. В свойстве `Address` этого компонента установите IP-адрес компьютера, на котором размещен сервер `MidasServer`. Это может быть другой компьютер в сети или тот же, на котором планируется установить клиентское приложение (в последнем случае IP-адресом будет `localhost`). Установите значение свойства

ServerName точно так же, как и при настройке компонента TDCOMConnection — откройте список ServerName и выберите в нем элемент MidasServer.CustomerOrders. Теперь проверьте, организовано ли соединение с сервером, — установите значение **true** в свойстве Connected компонента TSocketConnection. Но, как уже было сказано, после проверки нужно вернуть этому свойству значение **false**.

После этого установите в экранную форму еще одну кнопку (компонент TButton), назовите этот элемент управления **ButtonSocket**, а свойству Caption присвойте значение **Socket**. Текст обработчика события OnClick этого компонента представлен в листинге 19.9.

Листинг 19.9. Организация подключения к удаленному компьютеру

```
void __fastcall TForm1::ConnectTCPIP1Click(TObject *Sender)
{
  AnsiString S;
  if (InputQuery("Enter Machine Name or IP-Address:",
                 "Machine Name/IP-Address", S))
  {
    SocketConnection1->Address = S;
    ClientDataSet1->RemoteServer = SocketConnection1;
    ClientDataSet1->Active = true;
  }
}
```

После щелчка на кнопке Socket появится приглашение ввести IP-адрес удаленного компьютера, на котором находится сервер. Если система, которой вы пользуетесь, правильно настроена, можно ввести и "читабельный" эквивалент IP-адреса, например **localhost** или, как в моем случае, — **DrBob42**.

В приведенном выше обработчике события устанавливается значение свойства Address компонента TSocketConnection в соответствии с тем, что ввел пользователь в ответ на запрос. Затем устанавливается значение свойства RemoteServer компонента TClientDataSet — теперь этот компонент связывается с компонентом TSocketConnection. Последняя настройка — присвоение свойству Active компонента TClientDataSet значения **true**. При этом автоматически будет установлено значение **true** и в свойстве TSocketConnection.Connected.

На этой стадии разработки клиентское приложение соединено с сервером, и в его окне должны быть выведены переданные сервером данные. Описанную методику можно применять и для соединения с удаленным сервером, и для соединения с сервером, который работает на том же компьютере, что и клиентское приложение. Более того, вам не нужен NT-сервер домена, хотя я и рекомендую использовать его в приложениях на основе MIDAS.

Еще раз обращаю ваше внимание на то, что сервер MIDAS мы вообще не трогали. В нем не нужно ничего менять при переходе на другой протокол обмена. Фактически, если в вашем распоряжении имеется работающее серверное приложение MIDAS, то остается только зарегистрировать его в системном реестре — сервер сделает это самостоятельно при первом запуске после установки на новом компьютере. Учтите, что после перемещения сервера в другой каталог на том же компьютере регистрацию нужно повторить.

Обычно при создании MIDAS-приложений я предпочитаю пользоваться протоколом DCOM. Однако в некоторых случаях может понадобиться и протокол TCP/IP. Обычный RDM-модуль может общаться с клиентами и через Winsock, если соответствующий режим был разрешен при выполнении метода UpdateRegistry(). Клиенты должны подключаться к RDM-модулю с помощью компонента SocketConnection. Но чтобы подключение произошло, на компьютере сер-

вера MIDAS должен работать Winsock-сервер. В комплект C++Builder 4 входили две программы Winsock-сервера: `ScktSrvr.exe` (обычный Winsock-сервер) и `ScktSrvc.exe` (вариант Winsock-сервера в качестве службы NT). В комплекте C++Builder 5 имеется только один Winsock-сервер — `ScktSrvr.exe`, в котором объединены возможности обоих прежних вариантов — его можно запускать и как отдельное приложение, и как службу NT.

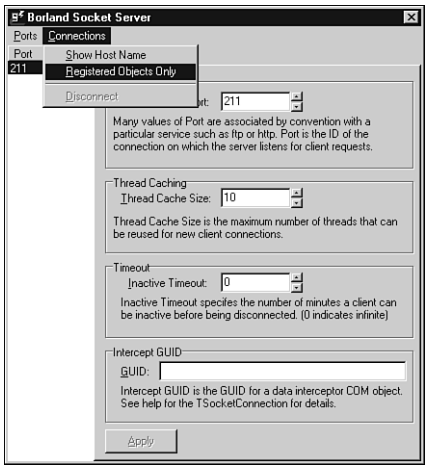


Рис. 19.24. Выбор режима работы Winsock-сервера

Ранее я уже говорил о том, что программа `ScktSrvr` просматривает реестр и проверяет, разрешено ли серверу MIDAS использовать Winsock при соединении с клиентом, т.е. было ли при выполнении метода `UpdateRegistry()` установлено значение **`true`** в поле `EnableSocket` объекта `regObj`. В серверах MIDAS, созданных в среде C++Builder 4, такая установка отсутствует, а потому при обновлении такого рода приложений, созданных в версии 4, нужно не забыть изменить программный код метода `UpdateRegistry()`. Если по каким-либо причинам новый сервер MIDAS не регистрирует себя в реестре как сервер, способный использовать Winsock, то нужно воспользоваться Winsock-сервером, поставляемым в комплекте с C++Builder 5 (но не тем, который идет в комплекте версии 4 C++Builder), и в меню Connections указать, что вы не собираетесь использовать режим Registered Object Only (Только зарегистрированные объекты) (рис. 19.24). Это позволит увидеть в списке и незарегистрированные объекты.

Новая настройка будет воспринята только после следующего запуска Winsock-сервера, но затем вы сможете увидеть все серверы MIDAS, созданные в среде C++Builder 5, которые используют компонент `TSocketConnection`, независимо от того, зарегистрировано ли это использование в системном реестре.

Еще одна новинка в компоненте `TSocketConnection` касается обратных вызовов. Компонент `TDCOMConnection` всегда поддерживает обратные вызовы, компонент `TWebConnection` эти вызовы не поддерживает вовсе, а компонент `TSocketConnection` можно настроить и так, и этак. Для настройки режима поддержки обратных вызовов в структуре компонента `TSocketConnection` предусмотрено свойство `SupportCallbacks`. Если вы считаете излишним маршалинг вызовов от сервера `MidasServer` к клиентскому приложению `MidasClient` через интерфейс обратного вызова, установите в свойстве `SupportCallbacks` значение **`false`** (по умолчанию устанавливается значение **`true`**). Установка такого режима позволяет при установке клиентского приложения MIDAS ограничиться версией сервера Winsock 1, в противном случае (когда `SupportCallbacks` имеет значение **`true`**) придется использовать Winsock 2 или более старшие версии. Поскольку в базовом комплекте операционной системы Windows 95 отсутствует Winsock 2, такая настройка несколько расширяет круг операционных систем, в которых возможна установка клиентских приложений MIDAS.

Брокер в выборе объектов

Выше мы рассмотрели функциональные возможности трех компонентов подключения, которые поставляются в комплекте программного продукта C++Builder 5 Enterprise: `TDCOMConnection`, `TWebConnection` и `TSocketConnection`. Все эти компоненты организуют со-

единение клиентов с одним серверным приложением MIDAS. Однако в некоторых случаях используется несколько однотипных серверов MIDAS. Для того бывают разные причины, в частности желание нивелировать загрузку ресурсов сети или повысить надежность работы приложения (если один из серверов перестанет функционировать, остальные возьмут на себя его функции).

А теперь задумайтесь над тем, каким образом можно на стороне клиента выбрать тот сервер, к которому следует подключиться. Для этого нужно располагать точной информацией о серверах, доступных в текущий момент. Но в MIDAS 3 можно избавить клиентское приложение от этих "забот", поскольку в этой версии реализована концепция *Object Brokering* (*Брокер в выборе объектов*).

Пользуясь средствами поддержки Object Brokering, клиентское приложение может подключиться к одному из однотипных серверов, даже "не зная" заранее, к какому именно. В структуре каждого из трех перечисленных выше компонентов подключения имеется свойство `ObjectBroker`. Это свойство используется для подключения к объекту класса, производного от `TCustomObjectBroker`. Этот объект затем передаст компоненту подключения информацию о том, какой из доступных однотипных серверов нужно использовать, установив значения свойств `ServerName` или `ServerGUID`. Обращаю ваше внимание на то, что при использовании `ObjectBroker` прежние значения `ServerName` и `ServerGUID` будут попросту игнорироваться. Заполнение этих свойств новыми значениями выполняется компонентом `ObjectBroker` динамически, в процессе работы клиентского приложения.

Как пример реализации концепции Object Brokering в комплект C++Builder 5 Enterprise включен компонент `TSimpleObjectBroker`, который находится на вкладке Midas палитры компонентов.

В структуре компонента `TSimpleObjectBroker` имеется два свойства, на которых я хочу остановить ваше внимание. В свойстве `Servers` хранится список доступных серверов. Каждый из них специфицируется записью, состоящей из следующих полей: `ComputerName` (системное имя компьютера в сети), `Port` (по умолчанию 211) и `Enabled` (флагом разрешения). Разработчик приложения должен заполнить этот список и обслужить его в процессе эксплуатации. При установке в системе нового сервера или удалении одного из серверов нужно обновить список `Servers`.

Свойство `LoadBalanced` имеет отношение к механизму распределения загрузки серверов. В основании этого механизма лежит генератор случайных чисел. При установке значения **`true`** в свойстве `LoadBalanced` сервер, к которому подключается очередное клиентское приложение, выбирается из списка случайно. При установке значения **`false`** в свойстве `LoadBalanced` (такое значение устанавливается по умолчанию), каждый очередной клиент подключается к тому серверу, который стоит в списке первым.

Компонент `TSimpleObjectBroker` реализован в модуле `ObjBrkr` (файлы этого модуля находятся в каталоге `$(BCB)\Source\Vcl`). Алгоритм работы компонента достаточно прост — случайный выбор сервера из списка, конечно же не очень оригинальная идея. Если вам придет в голову что-нибудь поинтереснее и вы решите разработать собственный компонент брокера, начните с модификации `TSimpleObjectBroker`.

Заключительное замечание относительно посредничества в выборе сервера. Текущее подключение к определенному серверу сохраняется до тех пор, пока в свойстве `Connected` не будет установлено значение **`false`**. При организации нового подключения (свойству `Connected` снова присваивается значение **`true`**) процесс выбора повторится, и в результате клиентское приложение может подключиться к совсем другому серверу.

Установка MIDAS-приложения

Установка разработанного MIDAS-приложения не требует особых усилий. Нужно отобрать DLL-модули и пакеты, необходимые для разработанного клиентского приложения, и включить в состав этого набора DLL-модуль поддержки MIDAS (модуль `MIDAS.DLL` для MIDAS 3 или модуль `DBCLIENT.DLL` для MIDAS 2). Никаких дополнительных драйверов баз данных не требуется. На компьютере клиента нужно зарегистрировать используемый сервер MIDAS (или серверы, если предполагается использовать несколько однотипных серверов с помощью посредника). Можно ограничиться и регистрацией только библиотеки типов сервера (об этом подробно рассказано в разделе *Удаленный доступ к серверу* в этой же главе).

Не забудьте о том, что необходимо приобрести официальную лицензию на право использования MIDAS. Лицензия на версию MIDAS 2 стоит достаточно дорого — US$5 000 на каждый компьютер, на котором будут установлены серверы MIDAS 2. Правда, количество установленных на этом компьютере серверов не ограничивается. В отношении MIDAS 3 действует другая модель лицензирования, которая, правда, не распространяется на обновлении версии MIDAS 2. Лицензия на использование MIDAS 3 при неограниченном количестве серверов обойдется вам всего в US$299,95. Кроме того, Borland впредь отказывается от лицензирования каждой клиентской инсталляции MIDAS 3.

В каких случаях нужно приобретать лицензию на право использования MIDAS? Это зависит от пакета данных MIDAS (переданного от провайдера к `ClientDataSet` или `XMLBroker` и обратно). В статье, касающейся политики лицензирования MIDAS 3, на сайте сообщества Borland Джон Кастер (John Kaster) сформулировал два правила.

1. Если пакет данных MIDAS передается любым способом с одного компьютера на другой, требуется приобретение лицензии.
2. Если пакет данных MIDAS остается на одном компьютере, приобретать лицензию не нужно.

Учтите, что под “любым способом” подразумевается и копирование на дискету, и передача по электронной почте, и резервное копирование на одном компьютере, а восстановление — на другом и прочие подобные методы.

Таким образом, затраты на лицензирование MIDAS 3 значительно снизились, что облегчило жизнь разработчикам, использующим в своей практике C++Builder 5.

Резюме

В этой главе была рассмотрена технология создания многоуровневых приложений на основе MIDAS. В частности, вы познакомились с методикой разработки серверных и клиентских приложений и использованием протоколов DCOM, TCP/IP и HTTP для соединения с удаленным сервером.

Для многих пользователей технология на основе MIDAS может полностью заменить прежнюю технологию создания приложений типа “клиент/сервер”. Для таких пользователей очень привлекательной является возможность разделения приложения на логически законченный компоненты, пусть даже и функционирующие на одном и том же компьютере. Но, естественно, наибольший выигрыш дает применение технологии на основе MIDAS в распределенной многомашинной среде.

Конечно, в этой главе мы не могли охватить всех аспектов применения MIDAS, но изложенное позволит вам приступить к более глубокому изучению этой технологии.

Глава

20

Распределенные приложения на основе модели CORBA

Рууд Ф. Пелс

Аббревиатура CORBA означает Common Object Request Broker Architecture (Архитектура универсального брокера запросов объектов). CORBA — это спецификация, разработанная группой поддержки объектной технологии (Object Management Group, или OMG). При ее разработке ставилась цель создать объектно-ориентированную архитектуру, независимую от платформы и допускающую масштабирование приложений. Средства поддержки архитектуры CORBA включены только в комплект программного продукта C++Builder 5 Enterprise.

В этой главе мы обсудим, что представляет собой архитектура CORBA, почему желательно ориентироваться на эту архитектуру при создании приложений, какие компоненты входят в состав средств поддержки CORBA в C++Builder 5. Мы затронем также и язык описания интерфейсов IML (Interface Definition Language), который используется при спецификации интерфейсов между подсистемами.

Затем перейдем к рассмотрению более практических вопросов — методики использования мастеров C++Builder, созданных для программирования CORBA-приложений. Вслед за этим мы обсудим модели реализации распределенных приложений на основе архитектуры CORBA.

Введение в CORBA

Одним из основных факторов при разработке программного обеспечения является размер системы. В последнее время размеры систем разрастаются как на дрожжах. Существующие инструментальные среды разработки позволяют создавать приложения, включающие порядка 10–30 миллионов операторов (строк программного кода). Еще десять лет назад программная система, включающая до одного миллиона операторов, считалась пределом возможного. Экстраполируя этот рост, можно смело прогнозировать, что к 2010 будут разработаны программные системы объемом свыше 100 миллиона операторов.

Потенциальная опасность такого разрастания объемов заключается в том, что вероятность их успешного тестирования, внедрения и сопровождения обратно пропорциональна размеру системы. Таким образом, успешное внедрение подобных сверхбольших систем возможно только при условии их разделения на подсистемы, взаимодействующие через четко специфицированные интерфейсы.

Одно из решений этой проблемы и предлагает архитектура CORBA. Во время подготовки этой книги к печати в качестве стандарта разработчики использовали спецификацию CORBA версии 2.3. Информация об этой спецификации доступна в Internet по адресу `http://www.omg.org`. Однако группа OMG занимается только разработкой стандартов, а реализация соответствующих программных средств поддержки этих стандартов отдана на откуп множеству независимых компаний. В настоящее время разработано множество средств поддержки CORBA на различных платформах, как программных, так и аппаратных. Кроме того, средства поддержки CORBA интегрированы в некоторые программные продукты, в частности СУБД Oracle 8i, систему извлечения документов Lotus Domino, менеджер окон X Windows (систему Gnome). Одно из главных преимуществ CORBA перед моделью составных объектов COM состоит в том, что распространением идей CORBA занято множество больших компаний, входящих в состав OMG. CORBA является *открытым стандартом*, и любой разработчик может реализовать его на основе той аппаратной и программной платформы, которая для него предпочтительна.

В C++Builder использованы средства реализации стандарта CORBA, созданные Visigenic, дочерней компанией Inprise.

Основы CORBA

В подавляющем большинстве реализаций CORBA (если не во всех) используется протокол TCP/IP. Для разработки интерфейсов программисты используют язык спецификации интерфейсов IDL (Interface Dafinition Langage). С помощью инструментальных средств поддержки CORBA созданная на этом языке спецификация интерфейса преобразуется в заготовку программного кода. Затем программист заполняет эту заготовку содержательным кодом. В большинстве случаев инструментальные средства поддержки CORBA не формируют при этом программный код на языке C++ в полном соответствии с последними стандартами этого языка по очень простой причине — не все компиляторы C++ способны работать с таким кодом.

В заготовке аргументы, передаваемые интерфейсом, преобразуются в такую форму, которая может быть интерпретирована на разных платформах. Это преобразование получило наименование *маршалинга* (*marshaling*) . На принимающем конце созданная заготовка восстанавливает аргументы и вызывает соответствующие методы (разработкой реализации этих методов занимается прикладной программист). Этот процесс получил наименование *демаршалинга* (*unmarshaling*). Затем весь этот механизм, включающий маршалинг на стороне сервера и демаршалинг на стороне клиента, используется вновь для передачи результатов клиенту. Это означает, что программа работающая на компьютере с процессором фирмы Intel, может вызывать метод в программе, работающей на компьютере с совершенно другой архитектурой. Код в созданной заготовке интерфейса и средства, входящие в уровень поддержки CORBA и обслуживания сети, выполняют все промежуточные преобразования.

Стандарт CORBA предполагает использование синхронного обмена. Это означает, что, если вызван метод удаленного объекта, то нужно выждать, пока ваш приятель не завершит работу. Но в этом стандарте имеется возможность и по-другому организовать взаимодействие, в частности использовать события и *однонаправленные* (one-way) вызовы.

Статика и динамика

Поскольку средства поддержки CORBA преобразуют вызовы методов в сообщения, то можно построить механизм обращения к методам CORBA с “чистого листа”. Это называется *динамическим обращением* — созданный код заготовки игнорируется, а сообщение формирует программа, которую программист разрабатывает самостоятельно. Но учтите, что при этом на программиста возлагается вся ответственность за корректную “сборку” сообщения, которое передается от клиента к серверу, и за “распаковку” ответа. При изменении определения интерфейса необходимо соответственно скорректировать и программный код. В этом смысле *статическое* обращение является более гибким, поскольку для проверки количества и типов аргументов можно воспользоваться возможностями компилятора.

Всегда или по требованию

В большинстве вариантов реализации средств поддержки CORBA предусмотрено два способа запуска сервера. Более простой — предполагает однократный запуск сервера, который в дальнейшем обслуживает поступающие от клиентов запросы. Другой способ предполагает использование так называемых *демонов активизации* (activation daemon). Такой демон представляет собой самостоятельный сервер, который интерпретирует поступающие запросы и отыскивает сервер, который сможет эти запросы обработать. Демон должен располагать информацией об интерфейсе, который реализован в активизируемом сервере, и о том, где найти этот сервер. Располагая этой информацией, демон самостоятельно запускает сервер в работу.

Одноуровневая или иерархическая организация

В стандарте CORBA предусмотрена специальная *служба формирования имен* (naming service). Она позволяет разработчику структурировать создаваемые средства так, как он посчитает нужным, в частности в виде иерархического дерева. Это имеет смысл при большом количестве создаваемых служб. Служба формирования имен позволяет связать логическое имя со службой, создаваемой динамически во время выполнения. Присвоение службе другого имени не потребует повторной компиляции программного кода. Служба формирования имен позволяет связать с одной динамически создаваемой службой разные имена. Такую процедуру организовать сложнее, но зато потом такие службы удобнее использовать.

Кто — сервер и кто — клиент

Стандарт CORBA не включает таких понятий как сервер и клиент. Любой процесс может играть роль и сервера, и клиента, или того и другого. Это означает, что программист может разработать программу клиентского процесса, которая в то же время будет работать и как серверный процесс.

Пусть, например, имеется приложение, которое должно получать уведомление об определенных событиях. Следовательно, сервер должен уведомлять только тех клиентов, которые прислали запрос с просьбой рассылать им уведомления. Это можно обеспечить, передав серверу ссылку на объект клиента. А это означает, что реализация уведомляющего объекта размещается на стороне клиента, а не в серверной части приложения.

Брокер запросов объектов

Ядром архитектуры CORBA является *брокер запросов объектов* (Object Request Broker — ORB). ORB-брокер обеспечивает соблюдение стандартов CORBA и является своего рода связующим звеном, объединяющим все компоненты приложения. Взаимодействие между клиентом и сервером осуществляется через ORB-брокер, причем таким образом, чтобы вызовы методов и их параметры можно было разрешить в адресном пространстве вызывающей или вызываемой процедуры. В C++Builder используется ORB-брокер VisiBroker, который реализован в виде модулей DLL. На каждом компьютере, имеющем клиентское и серверное приложение CORBA, должен быть установлен модуль ORB-брокера.

Адаптер базисных объектов

Адаптер базисных объектов (Basic Object Adapter, или BOA) представляет в технологии CORBA механизм связи между ORB-брокером и серверами. BOA регистрирует службы, поддерживаемые сервером, т.е. позволяет ORB-брокеру отыскать службы, которые запрашиваются клиентом.

CORBA или COM?

Сравнивая модель COM, во всех ее вариациях, существующих на сегодняшний день, и архитектуру CORBA, можно выявить как преимущества первой, так и недостатки. К преимуществам COM следует отнести следующие:

- средства поддержки COM входят в состав операционной системы и дополнительных затрат на их приобретение не требуется;
- модель COM получила за последнее время очень широкое распространение.

Недостатки модели COM:

- модель поддерживается только на некоторых платформах;
- COM не является открытым стандартом;
- клиенты DCOM должны располагать выполняемым файлом сервера для того, чтобы организовать его регистрацию (в противном случае не удастся вызвать сервер со стороны клиента);
- модель COM плохо документирована.

Если разрабатываемые вами программные продукты ориентированы в основном на архитектуру WinTel, то следует отдать предпочтение использованию COM, DCOM или COM+. В такой инфраструктуре эти модели работаю прекрасно. Но если в приложение должны входить компоненты, работающие на совершенно разных аппаратных и программных платформах, то модель COM и ее варианты помочь вам не смогут. Хуже того, в средах разработки, поставляемых Microsoft, отсутствуют средства поддержки архитектуры CORBA. Приверженцам Microsoft придется воспользоваться подходом, который я называю "CORBA для бедных" (о нем речь в конце этой главы).

Компоненты *Visibroker*

Помимо средств формирования заготовок интерфейсов для C++, клиентов Delphi CORBA и Java, в комплект средств поддержки ***Visibroker*** входит три основных компонента. Это не единственные компоненты в этом комплекте, но для создания системы, базирующейся на CORBA, их вполне достаточно. Перечень этих компонентов включает:

- службу Smart Agent;
- демон активизации объектов Object Activation Daemon;
- компонент Console.

Служба Smart Agent

Выполняемый файл `osagent` принято называть *службой Smart Agent*. Эта служба не является частью спецификации CORBA, но играет очень важную роль при реализации этой архитектуры — наподобие той, которую играет телефонный справочник в нашей повседневной жизни. Как только сервер CORBA запускается в работу, он обращается к службе Smart Agent и просит ее зарегистрировать информацию об интерфейсах, которыми он располагает. Клиентская программа CORBA, в свою очередь, обращается к той же службе Smart Agent и просит ее отыскать необходимый ей интерфейс. Она также просит агента соединить клиента с сервером. Таким образом, служба Smart Agent действительно играет роль агента-посредника между сервером CORBA и клиентом CORBA. После того как между ними установлено соединение, агент может "отдыхать".

Демон активизации объектов

Наш "сообразительный" агент в определенном смысле немного туповат. Он полагает, что если интерфейс доступен, то, разумеется, сервер работает. Если на компьютере сервера используется *демон активизации объектов* (Object Activation Daemon, или OAD), то эта ошибка вполне простительна. Демон активизации объектов располагает достаточной информацией для того, чтобы запустить серверный процесс после того как агент запросит соединение кли-

ента с этим процессом. Эта информация — сам интерфейс, имя сервера и все необходимые аргументы и переменные среды —сохраняется в хранилище интерфейсов (Interface Repository, или IREP). Занесение этой информации производится инструментальной программой `oadutil`. Лучше всего обращаться к ней из пакетного файла.

Утилита `Console`

Этот компонент — новинка Visibroker 4. В прежних версиях Visibroker компонент `Console` отсутствовал. Эта утилита позволяет просмотреть список доступных служб CORBA.

Язык определения интерфейсов

CORBA-версия языка определения интерфейсов (Interface Definition Language, или IDL) не слишком отличается от версии Microsoft, которая используется для определения COM-интерфейсов. Ее часто называют Microsoft IDL, а программу преобразования описания интерфейса — `MIDL`. В программе `MIDL` используются файлы определения COM-интерфейсов с расширением `.idl`.

Язык IDL позволяет специфицировать интерфейс — его свойства и методы, но не дает никакой информации о реализации заявленных методов интерфейса. Реализация — забота программиста, и для этого используются языки C++, Java и им подобные. На языке IDL идентификатор может содержать буквы, цифры и символы подчеркивания. Правда, последние использовать не рекомендуется, поскольку некоторые языки программирования не позволяют включать этот символ в идентификатор. В определение интерфейса на языке IDL можно включать и директивы препроцессора, поскольку компилятор IDL оснащен собственным препроцессором.

Язык IDL позволяет использовать директиву `typedef` для определения собственного типа. В качестве основных типов поддерживаются `float`, `double`, `long`, `short`, `unsigned long`, `unsigned short`, `char`, `boolean`, `octet`, `Any` и `NamedValue`. Тип `Any` в IDL играет ту же роль, что и тип `Variant` в COM. Составной тип `NamedValue` включает *имя* (строку) и *значение* типа `Any`. Язык IDL также поддерживает многомерные массивы с заданной размерностью. В нем имеется два типа шаблонов: `sequence` и `string`. Шаблон `sequence` представляет собой одномерный массив переменных определенного типа, либо ограниченный (`typedef sequence<octet,6> SixBytesMax`), либо неограниченный (`typedef sequence<any> ABunchOfAnyes`). Шаблон `string` — это последовательность переменных типа `char`. Учтите, что при использовании строк ограниченной длины замыкающий нуль не входит в длину строки, т.е. `typedef string<15>` — это строка именно из 15 символов, а не из 14 и замыкающего нуля. Строкам IDL соответствует тип `char*` в C++. Для динамического выделения памяти под строку нужно использовать операторы `CORBA::string_alloc(CORBA::Ulong)` и `CORBA::string_free(char*)`.

Язык IDL также поддерживает и структурные типы `struct`, `union` и `enum`. Типы `struct` и `enum` аналогичны (в смысле синтаксиса) своим “собратьям” в C++, а тип `union` выглядит несколько по-другому. В определение включено поле тэга, которое определяет, какой член `union` является текущим.

```
union ExampleUnion switch(long)
{
  case 1: long x;
  case 2: string y;
} ;
```

Поддерживаются и константы. Синтаксис определения констант выглядит точно так же, как в C++. Например:

```
const long Foo = 42;
```

Язык IDL не поддерживает констант типа `octet`.

Ключевое слово `interface`

Интерфейс объявляется с ключевым словом `interface`. В объявлении интерфейса указывается, какие "услуги" сервер может предоставить своим клиентам. В определении не предусмотрено ограничение доступа к членам, т.е. подразумевается, что все члены интерфейса являются общедоступными (public). CORBA-интерфейс соответствует классу C++, и все сгенерированные классы являются производными от `CORBA::Object`. Можно использовать опережающее объявление интерфейса для ссылки на него. Можно даже ссылаться в определении интерфейса на самого себя.

Ключевое слово `attribute`

Атрибуты в языке IDL играют ту же роль, что и свойства в Delphi или C++Builder. Если необходимо запретить клиенту записывать что-либо в атрибут, следует его определение предварить ключевым словом `readonly`:

```
interface ReadWrite {
  attribute long CanBeWritten;
  readonly attribute string ReadOnly;
}
```

В сгенерированной на основании этого определения заготовке на языке C++ в атрибут будет включена функция считывания значения `get..` и, если в определении отсутствует ключевое слово `readonly`, — функция установки `set..` (в имена обоих функций входит и имя атрибута).

Методы

Синтаксис объявления методов интерфейса в IDL полностью повторяет синтаксис объявления методов (членов-функций) в языке C++. Нужно задать тип возвращаемого значения, а каждый аргумент метода должен сопровождаться ключевым словом, задающим направление передачи этого аргумента. Если аргумент передается *серверу*, он должен сопровождаться ключевым словом `in`, если аргумент передается от *сервера к клиенту*, он должен сопровождаться ключевым словом `out`. Есть и третий вариант — аргумент передается в обоих направлениях — в этом случает он должен сопровождаться ключевым словом `inout`. Если не планируется ожидать результат вызова метода, его возвращаемое значение должно иметь тип `void`, а в конце строки объявления метода нужно поместить ключевое слово `oneway`.

Определение типов

В определение интерфейса может быть включено и определение типа. Поскольку отдельный интерфейс определяет отдельную зону видимости имен, то ссылка на такой тип извне интерфейса должна содержать оператор задания зоны видимости, например `MyInterface::MyTypeDefinition`.

Исключения

В интерфейс можно включать и типы исключений. При реализации на языке C++ все классы исключений формируются как производные от класса `CORBA::UserException`, который, в свою очередь, является производным от класса `CORBA::Exception`. Типы исключений можно использовать в спецификации методов, чтобы указать, какие из них должны быть сгенерированы при вызове данного метода:

```
interface Pitcher {
  exception NoBallAvailable {
  } ;
  void Throw() raises (NoBallAvailable);
} ;
```

Для возбуждения исключения в реализации объекта CORBA на языке C++ используется ключевое слово `throw`:

```
void PitcherImpl::Throw() {
  if (IcannotFindABall() == true) {
    throw Pitcher::NoBallAvailable;
  }
} ;
```

Перехват исключения, специфицированного в CORBA-интерфейсе, выполняется так же, как и исключения любого другого типа:

```
void PitcherClient::Throw() {
  try {
    pitcher.Throw();
  }
  catch (Pitcher::NoBallAvailable& pitcherx) {
    // Перехват исключения Pitcher::NoBallAvailable
    ...
  }
  catch (CORBA::UserException& userx) {
    // Перехват всех прочих исключений CORBA.
    ...
}
```

Наследование

В CORBA поддерживается наследование, в том числе и множественное. Но в случае множественного наследования нужно иметь в виду, что интерфейсы, от которых наследуется данный, не должны иметь членов с совпадающими именами. В интерфейсе-наследнике не нужно переобъявлять метод, объявленный в интерфейсе-родителе.

Все интерфейсы CORBA неявно наследуют интерфейс `CORBA::Object`. Это означает, что передавая параметр типа `CORBA::Object`, можно передавать объект любого CORBA-интерфейса.

Модули

В модуле интерфейсы группируются примерно так же, как в пространстве имен (namespace) C++. При правильной настройке ключей компилятор IDL формирует программный код, в котором отдельный модуль отображается на отдельное пространство имен. Мо-

дуль ограничивает зону видимости имен. Для того чтобы извне обратиться к имени, определенному внутри модуля, нужно воспользоваться оператором явного определения зоны видимости, например `MyModule::MyInterface`.

Новинки C++Builder 5

В новой версии C++Builder мало что изменилось в средствах поддержки архитектуры CORBA. Мастера C++Builder работают с устаревшим стандартом CORBA, который поддерживается Visibroker 4.

Но в новой версии Visibroker изменилось практически все. Например, ниже показано, как ранее запускался сервер CORBA:

```
CORBA::ORB_var orb = CORBA::ORB_init(argc, argv);
CORBA::BOA_var boa = orb->BOA_init(argc, argv);
BarImpl bar_BarObject("BarObject");
boa->obj_is_ready(&bar_BarObject);
boa->impl_is_ready();
```

А вот как это выполняется в соответствии с новой спецификацией CORBA:

```
CORBA::ORB_var orb = CORBA::ORB_init(argc, argv);
CORBA::Object_var obj =
  orb->resolve_initial_references("RootPOA");
PortableServer::POA_var rootPOA =
  PortableServer::POA::_narrow(obj);
CORBA::PolicyList policies;
policies.length(1);
policies[(CORBA::ULong)0] =
  rootPOA->create_lifespan_policy(PortableServer::PERSISTENT);
PortableServer::POAManager_var poa_mgr =
  rootPOA->the_POAManager();
PortableServer::POA_var myPOA =
  rootPOA->create_POA("bar_poa", poa_mgr, policies);
BarImpl barServant;
PortableServer::ObjectId_var barId =
  PortableServer::string_to_ObjectId("Bar");
myPOA->activate_object_with_id(barId, &barServant);
poa_mgr->activate();
orb->run();
```

Можно ли это считать новинкой? Лично я думаю, что можно. Дело в том, что ранее группа разработчиков стандарта (группа OMG) не специфицировала многих вещей, касающихся адаптера базисных объектов BOA. В новом стандарте версии 2.3 уже имеется подробная спецификация, которая и реализована в Visibroker 4. Если вы заглянете в сопроводительную документацию Visibroker 4, то, несомненно, обратите внимание на значительное расширение функциональных возможностей новой версии, по сравнению с той, которая входила в комплект C++Builder 4.

Учтите, что некоторые новинки требуют к себе особого внимания. Во-первых, в языке IDL появилось довольно много новых ключевых слов, во-вторых, теперь в ключевых словах не различаются строчные и прописные буквы, т.е. впредь нельзя использовать имена, отличающиеся от ключевых слов только регистром букв. Коме того, имеется множество мелких

изменений в правилах формирования имен. Подробную информацию можно получить в сопроводительной документации Visibroker 4.

Впредь не будет поддерживаться модель `ptie`. Поскольку архитектура брокера запросов объектов ORB претерпела очень серьезные изменения, надобность в этой модели просто исчезла.

Поддержка технологии CORBA в среде C++Builder

Теперь перейдем к особенностям программирования приложений на базе архитектуры CORBA в среде разработки C++Builder. В последующих разделах мы "пройдемся" по всем элементам среды разработки, которые имеют отношение к созданию CORBA-приложений, и объясним их назначение.

Диалоговое окно Environment Options

Органы настройки окружения, связанные с программированием CORBA-приложений, размещены на вкладке CORBA диалогового окна Environment Options. Для вывода этого окна на экран нужно выбрать в главном меню C++Builder команду Tools⇨Environment Options. На вкладке имеется два флажка настройки (рис. 20.1).

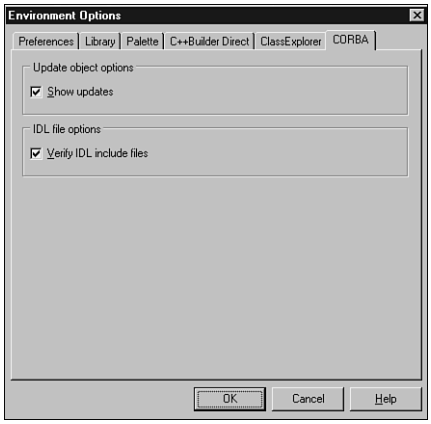


Рис. 20.1. Вкладка CORBA диалогового окна Environment Options

C++Builder располагает средствами обновления реализации CORBA-интерфейсов при изменении определения интерфейса на языке IDL. Флажок Show Updates (Показать обновления) управляет режимом выполнения обновлений с помощью C++Builder. Если флажок установлен, то при обнаружении изменений в определении интерфейса на языке IDL на экран выводится диалоговое окно Project Updates, с помощью которого можно вручную внести необходимые изменения в классы реализации интерфейсов. Если же флажок Show Updates сброшен, изменения будут внесены автоматически.

Если флажок Verify IDL Include Files (Проверять включение файлов IDL) установлен, то C++Builder будет выполнять поиск необходимых IDL-файлов, не включенных в проект. Если такой файл не включен в проект, описанные в нем интерфейсы не принимаются во внимание при компиляции проекта. Возможна ситуация, когда в одном из файлов проекта имеется ссылка на IDL-файл, а сам файл в состав проекта не включен. Именно в такой ситуации и нужна установка флажка Verify IDL Include Files. Если он установлен, C++Builder предложит пользователю добавить такой IDL-файл в состав проекта с помощью диалогового окна Add Included IDL Files (Добавление IDL-файлов).

Диалоговое окно Debugger Options

Органы настройки средств отладки, связанных с программированием CORBA-приложений, размещены на вкладке Language Exceptions (Исключения) диалогового окна Debugger Options. Для вывода этого окна на экран нужно выбрать в главном меню

C++Builder команду Tools⇨Debugger Options. На этой вкладке можно изменить установку трех флажков типов исключений, которые используются при создании CORBA-приложений — Visibroker Internal Exceptions, CORBA System Exceptions и CORBA User Exceptions (рис. 20.2). В списке имеется и еще два флажка — VCL EAbort Exceptions и Microsoft DAO Exceptions, но они к CORBA никакого отношения не имеют.

Исключения отмеченных типов будут игнорироваться отладчиком C++Builder.

Диалоговое окно Project Options

Органы настройки проекта, используемые при программировании CORBA-приложений, размещены на вкладке CORBA диалогового окна Project Options. Для вывода этого окна на экран нужно выбрать в главном меню C++Builder команду Tools⇨Project Options. На вкладке имеется несколько зон настройки разных групп параметров проекта (рис. 20.3). Эти опции определяют, каким образом компилятором IDL будет транслироваться определение интерфейса на языке IDL в программный код *заглушки* (*stub*) и что будет делать среда C++Builder с результатами трансляции.

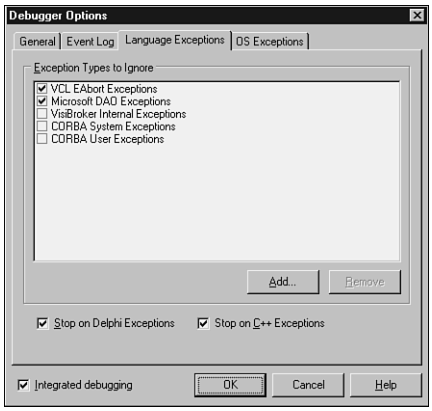


Рис. 20.2. Вкладка Language Exceptions диалогового окна Debugger Options

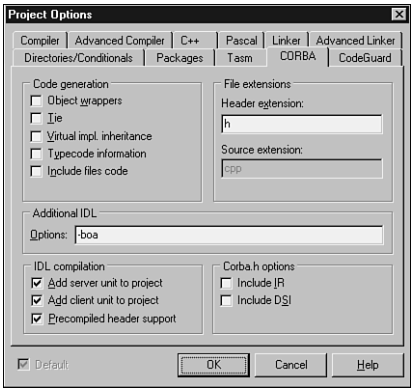


Рис. 20.3. Вкладка CORBA диалогового окна Project Options

В группе флажков Code Generation определяется тип генерируемого программного кода. Назначение флажков Tie и Virtual Impl. Inheritance будет рассмотрено в разделе *Модели реализации* этой главы. При установке флажка Object Wrapper генерируется код, который выполняет функции *оболочки* (*wrapper*) по отношению к заглушкам интерфейсов сервера и клиента. На стороне клиента методы оболочки вызываются до формирования вызова заглушки и после получения ответа от метода заглушки. На стороне сервера методы оболочки вызываются перед обращением к объекту реализации и после получения ответа от этого объекта. Объекты оболочки могут использоваться для кэширования результатов, отладки и операций временной синхронизации. Более подробную информацию об объектах оболочки можно получить в сопроводительной документации ***Visibroker***. При установленном флажке Typecode Information будет генерироваться программный код, необходимый для клиентского приложения, в котором используются *интерфейсы динамического вызова* (Dynamic Invocation Interface). Если этот флажок сброшен, то пользователь должен самостоятельно зарегистрировать сервер в хранилище интерфейсов ***Interface Repository*** демона активизации объектов OAD. Более подробную информацию о назначении этого флажка можно получить в сопроводительной документации ***Visibroker***.

Правее расположена зона File Extensions. Размещенные в ней элементы управления позволяют настроить расширение имен файлов заголовков, формируемых компилятором IDL.

В зоне Additional IDL можно указать дополнительные флаги вызова компилятора IDL. Не следует удалять предлагаемый в этом поле настройки флаг `-boa`; этот флаг позволяет компилятору `idl2cpp` генерировать имена классов. обладающие обратной совместимостью. Если этот флаг удалить, мастер C++Builder работать не сможет.

Слева внизу находится зона IDL Compilation, в которой задаются операции, выполняемые C++Builder после завершения компиляции IDL-файла. Справа от нее находится зона Corba.h Options, где можно указать дополнительные средства поддержки полученного программного кода заготовки интерфейса. Если флажок Include IR установлен, компилятор IDL добавит в модуль заготовки код, необходимый для работы с хранилищем интерфейсов ***Interface Repository***. Если установлен флажок Include DSI, компилятор IDL добавит в модуль заготовки программный код, который позволит использовать компонент ***Dynamic Skeleton Interface***. Более подробную информацию об этих настройках вы сможете найти в сопроводительной документации ***Visibroker***.

Мастер *CORBA Server Wizard*

Ярлык мастера ***CORBA Server Wizard*** находится на вкладке Multitier диалогового окна New Items, которое выводится на экран после выбора в главном меню команды File⇨New. Этот мастер (его диалоговое окно представлено на рис. 20.4) позволяет указать, будет ли сервер CORBA приложением Windows или консольным приложением.

Рис. 20.4. Окно мастера CORBA Server Wizard

В консольном приложении можно добавить поддержку VCL. В нижней половине окна мастера размещен список IDL-файлов, добавляемых в проект. Мастер позволяет также сформировать новый IDL-файл. Результатом работы мастера является новый проект. При создании консольного приложения обратите внимание на переменную `Cerr`. Она определена в одном из включаемых файлов ***Visibroker*** и позволяет сформировать программный код реализации CORBA переносимым с одного компилятора на другой. Ниже показано как это делается:

```
#if defined(_VIS_STD) && !defined(BUILD_ORBDLL) && !defined(_VIS_STD_NO_DEF)
#define Coutstd::cout
#define Cerrstd::cerr
#define Clogstd::clog
#define Endlstd::endl
#else
#define Coutcout
#define Cerrcerr
#define Clogclog
#define Endl      endl
#endif
```

Таким образом, с помощью директив препроцессора разработчики ***Visibroker*** организовали условную компиляцию, которая воспринимается как более старыми компиляторами, не поддерживающими пространств имен, так и более новыми.

Мастер *CORBA Client Wizard*

На вкладке Multitier диалогового окна New Items также находится ярлык мастера ***CORBA Client Wizard***. Окно этого мастера практически не отличается от диалогового окна мастера ***CORBA Server Wizard*** (рис. 20.5).

Отличие только в том, что с помощью этого мастера вы можете выбрать режим инициализации адаптера базисных объектов BOA, но не можете создать новый IDL-файл. Пользователю дозволено только добавлять уже существующие IDL-файлы в проект клиентского CORBA-приложения. Результатом работы мастера, как обычно, является новый проект.

Мастер *CORBA IDL File Wizard*

Ярлык этого мастера находится на все той же вкладке Multitier диалогового окна New Items. Окно этого мастера похоже на окна прочих мастеров создания файлов. Этот мастер формирует новый файл с расширением `.idl`.

Мастер *CORBA Object Implementation Wizard*

Если в состав текущего проекта не включен ни один IDL-файл, этот мастер сразу же прекратит работу. Прежде чем вызывать мастера ***CORBA Object Implementation Wizard***, нужно настроить переменную окружения `CLASSPATH` таким образом, чтобы компилятор IDL смог отыскать необходимый JAR-файл. При возникновении проблем вызовите программу `idl2cpp`, которая находится в подкаталоге `bin` каталога `vbroker`. В этом подкаталоге хранятся все выполняемые файлы ***Visibroker***.

После того как мастер ***CORBA Object Implementation Wizard*** запущен (его ярлык находится на вкладке Multitier диалогового окна New Items), он запускает на выполнение компилятор IDL. Процесс компиляции займет некоторое время, поэтому придется подождать. После завершения работы компилятора на экране откроется диалоговое окно мастера CORBA Object Implementation Wizard (рис. 20.6).

Рис. 20.5. Окно мастера CORBA Client Wizard

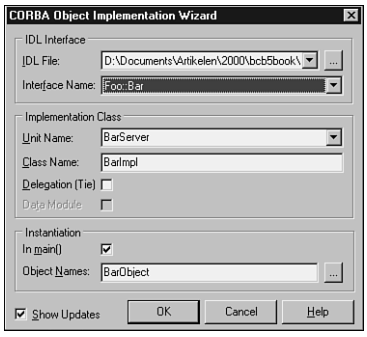


Рис. 20.6. Окно мастера CORBA Object Implementation Wizard

В списке IDL File зоны IDL Interface выберите имя одного из IDL-файлов из тех, что включены в состав проекта. В поле Interface Name укажите имя интерфейса, для которого будет создаваться класс реализации. В полях Unit Name и Class Name зоны Implementation Class нужно указать имя модуля и имя класса реализации интерфейса. Если создается серверное приложение, в котором разрешена поддержка VCL и установлен флажок Delegation

(Tie), то будет создан модуль данных, в который можно помещать необходимые для реализации CORBA-объекта компоненты.

В зоне Instantiation настраивается режим формирования экземпляра CORBA-объекта. При создании консольного приложения нужно установить флажок In main(), а при создании приложения Windows — сбросить его. В поле Object Names нужно указать одно или несколько имен объектов CORBA того класса. который задан в зоне Implementation Class.

Если на вкладке CORBA диалогового окна Environment Options установлен флажок Show Updates, то дальнейшие операции выполняются в диалоговом окне Project Updates.

Диалоговое окно Project Updates

На поле диалогового окна Project Updates размещены две панели (рис. 20.7).

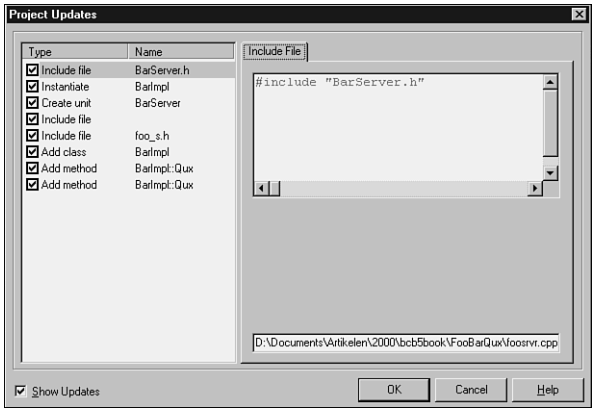


Рис. 20.7. Диалоговое окно Project Updates

В левой панели выведен список изменений, а в правой — детальная информация об этих изменениях. Если пользователь не согласен с одним или несколькими предлагаемыми C++Builder изменениями, он должен сбросить соответствующий флажок в левой панели. Надпись на корешке правой панели несет информацию о том, в какой компонент проекта предлагается внести изменения. Иногда таким компонентом является модуль, в других случаях — файл включения или блок программного кода. В некоторых случаях в правой панели появляется две вкладки: Definition (Определение) и Implementation (Реализация). Все зависит от характера изменений, предлагаемых средой C++Builder. При желании можно редактировать программный код в правой панели. В самом низу правой панели выводится имя файла, в который вносятся изменения. Это имя редактировать нельзя. После выполнения всех операций щелкните на кнопке OK.

Мастер *Use CORBA Object Wizard*

Обратиться к услугам этого мастера можно двумя способами: выбрать в главном меню C++Builder команду Edit⇨Use CORBA Object или вывести на экран панель инструментов CORBA (выбрав команду View⇨Toolbars⇨CORBA) и щелкнуть на пиктограмме Use CORBA Object Wizard — самой правой на этой панели инструментов. (На этой панели есть еще две пиктограммы: слева находится CORBA Object Implementation Wizard, а в центре — пиктограмма вызова диалогового окна Project Updates.) После выполнения любой из операций на экране появится окно мастера Use CORBA Object Wizard (рис. 20.8).

Этот мастер способен подготовить программный код, необходимый для использования CORBA-объекта клиентом. Элементы управления в зоне CORBA Object позволяют выбрать IDL-файл, в котором определен интерфейс используемого объекта. Выбрав один из интерфейсов в этом файле, нужно указать имя объекта. Это должно быть одно из тех имен, которые были указаны в поле Object Names окна мастера CORBA Object Implementation Wizard.

Ниже зоны CORBA Object находятся три вкладки. Первая из них — Use in Form.

На этой вкладке можно выбрать имя формы и наименование свойства. C++Builder формирует это свойство и фрагмент программного кода, обеспечивающий доступ к CORBA-объекту. При первом использовании сформированного свойства этот программный код связывает удаленный CORBA-объект с приложением.

Вторая вкладка — Use in main() — показана на рис. 20.9. В ней имеется только одно поле с именем переменной, посредством которой в модуле `main()` можно получить доступ к удаленному объекту.

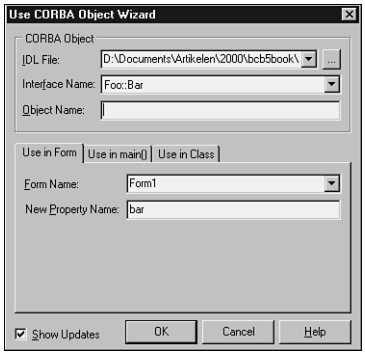


Рис. 20.8. Диалоговое окно мастера Use CORBA Object Wizard, в котором открыта вкладка Use in Form

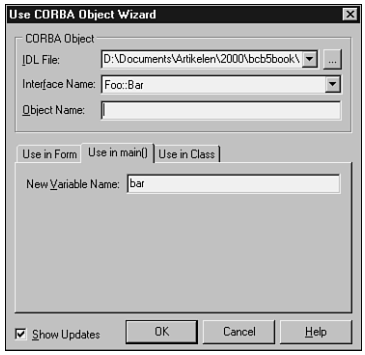


Рис. 20.9. Вкладка Use in main() диалогового окно мастера Use CORBA Object Wizard

Последняя вкладка — Use in Class — показана на рис. 20.10. Размещенные на ней элементы управления позволяют организовать класс оболочки для удаленного CORBA-объекта. Это довольно здравая идея, поскольку класс оболочки будет инкапсулировать все детали обращения к удаленному CORBA-объекту, и избавит от ненужных "подробностей" прочие компоненты приложения.

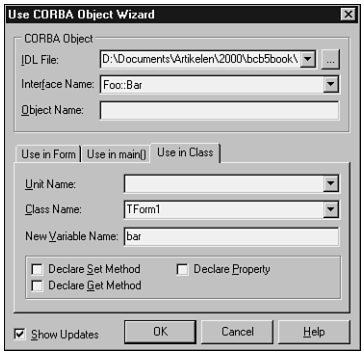


Рис. 20.10. Вкладка Use in Class диалогового окно мастера Use CORBA Object Wizard

Отличия между C++Builder 4 и C++Builder 5

С точки зрения среды разработки, существенных отличий между C++Builder версий 4 и 5 нет. Несколько изменилась компоновка некоторых диалоговых окон, добавлены кое-какие новые функции — и это все. Для ограничения зоны видимости в C++Builder по-прежнему используются классы, а не пространства имен. Что радикально изменилось, так это возможности новой версии ***Visibroker***. Версия 4 существенно отличается от той, что поставляется в комплекте с Delphi 5 и JBuilder 3. Учтите, что при установке C++Builder 5 после JBuilder 3 или Delphi 5 могут возникнуть проблемы с использованием ***Visibroker*** в двух последних средах разработки. Я бы рекомендовал вам установить версию JBuilder 3.5, в которой используется та же версия ***Visibroker***, что и в C++Builder 5. Пользователям Delphi 5 в этом отношении повезло меньше. Для того чтобы эксплуатировать Delphi 5, нужно вновь установить DLL-модуль `orb_br.dll` (это версия ***Visibroker***, которой комплектуется Delphi 5), переименовать его в `orbpbr.dll` и затем отредактировать в шестнадцатеричном коде модуль `orbpas50.dll` таким образом, чтобы он обращался к `orbpbr.dll` вместо `orb_br.dll`.

Модели реализации

В C++ существует множество вариантов выбора модели реализации, которые схематически представлены на рис. 20.11. Предположим, что разработанный интерфейс имеет наименование `Bar` и "прописан" в модуле `Foo`. Если просмотреть схему наследования интерфейсов, то окажется, что в эту схему вовлечено довольно много классов. Можно, например, найти *класс каркаса* (*skeleton class*), который является производным сразу от трех классов. Именно это происходит при использовании ключа `-boa` в строке вызова компилятора `idl2cpp`. Соответствующие компоненты на схеме (рис. 20.11) выделены утолщенными линиями, и именно они обеспечивают обратную совместимость.

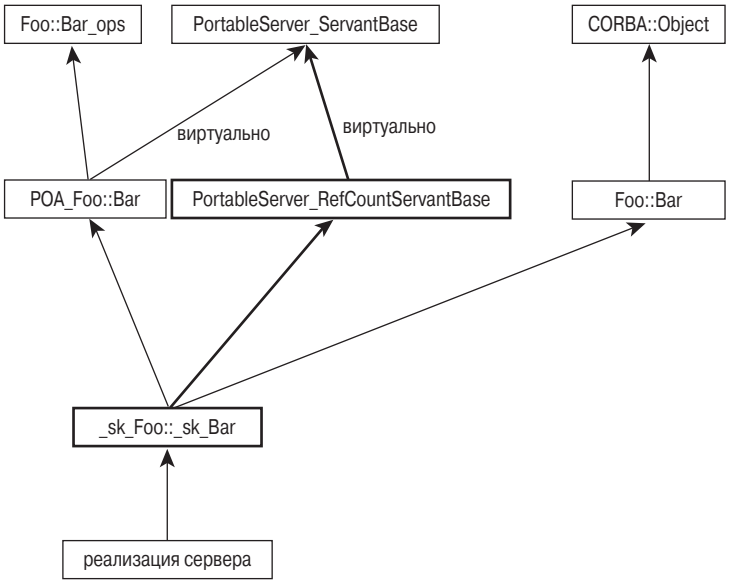


Рис. 20.11. Модель объекта, которая обеспечивает обратную совместимость

Если этот ключ при вызове компилятора не использовать, то объект сервера будет наследоваться от `POA_Foo::Bar`, а в клиентском приложении по-прежнему будет использоваться класс `Foo::Bar`. Приведенная на рис. 20.12 схема наследования значительно проще.

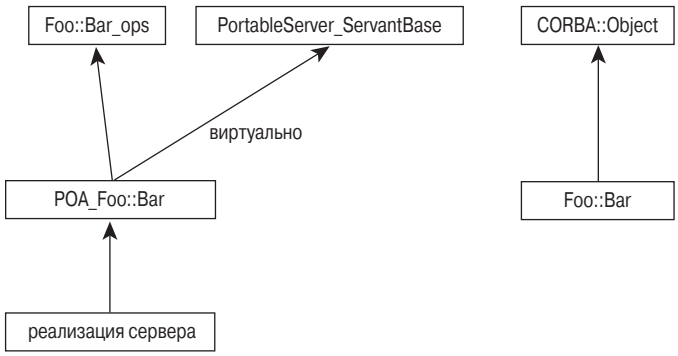


Рис. 20.12. Модель объекта в соответствии с новым стандартом CORBA

Но, увы, мастера C++Builder не смогут работать с таким вариантом реализации.

То, что вы видите на этой схеме, — новый стандарт CORBA в части реализации класса сервера, не наследующего класс `CORBA::Object`. Он является производным от `PortableServer_ServantBase`, а тот, в свою очередь, является виртуальным производным от `POA_Foo::Bar`. Именно такую схему наследования мы называем *смешанным классом* (mix-in class). Класс `POA_Foo::Bar` наследует методы реализации от класса `Foo::Bar_ops`, а CORBA-методы от класса `PortableServer_ServantBase`.

Наследование

Модель наследования является самой простой из существующих. Класс реализации нужно сделать производным от `_sk_Foo::_sk_Bar` (префикс `_sk_` означает *skeleton* — каркас). Класс каркаса, в свою очередь, является производным от класса интерфейса, который мы будем использовать на стороне клиента, а последний, в свою очередь, является производным от `CORBA::Object`. В новом стандарте CORBA следует организовать наследование непосредственно от `POA_Foo::Bar`. В обоих случая требуется разработать методы доступа и установки атрибутов, объявленных в структуре интерфейса.

Виртуальная реализация наследования

В этом случае единственное отличие состоит в том, что класс `Foo::Bar` виртуально наследует от `CORBA::Object`. Это может оказаться весьма кстати, если вы захотите включить CORBA-интерфейс в другую иерархию наследования на стороне клиента. Существует по крайней мере один пример виртуального наследования от `CORBA::Object`. Поскольку класс `Foo::Bar` является спецификацией интерфейса на стороне клиента, подумайте над тем, как использовать эти классы в качестве интерфейсов, трактуя понятие “интерфейс” так, как это принято в мире Java. Можно создать объект клиента, которому необходимы два или более клиентских CORBA-интерфейсов. Этот класс нужно сделать производным от `Foo::Bar` и `Foo:Fiz`. После этого нужно использовать виртуальный интерфейс для ссылки на тот же объ-

ект CORBA::Object в самом верху дерева наследования. Пусть клиентский объект для выполнения своих функций нуждается в двух объектах CORBA: Foo::Bar и Foo::Fiz. Если не использовать виртуального наследования, то Foo::Bar и Foo::Fiz будут ссылаться на два разных объекта CORBA::Object. Используя виртуальное наследование можно организовать ссылку на один и тот же объект CORBA::Object.

Модель делегирования

Если выбрана модель делегирования на стороне сервера (в зоне Implementation Class диалогового окна мастера CORBA Object Implementation Wizard установлен флажок Delegation (Tie)), то основные функции возлагаются на класс, который не имеет никакого отношения к CORBA. Будет использован шаблон класса, который содержит указатель на аргумент template. Любая операция переадресуется классу реализации через этот указатель. Но класс реализации должен предоставлять методы для всех операций, специфицированных в объявлении интерфейса на языке IDL. При создании экземпляра объекта сервера нужно предоставить ему класс template с указателем на класс. который в действительности будет реализовывать метод извлечения и установки атрибутов IDL-интерфейса.

Такая модель реализации CORBA-интерфейса выглядит очень элегантно, поскольку позволяет "развязать" в серверном приложении компоненты, связанные с CORBA, и компоненты, выполняющие основные функции сервера.

CORBA для „бедных"

Если в вашем распоряжении нет редакции Enterprise среды C++Builder, JBuilder или Delphi, то из всего, что было сказано выше, последует вывод: CORBA — не для нас. Но не спешите — вы все-таки можете использовать идеи архитектуры CORBA. На ваше счастье остаются еще средства поддержки CORBA в среде разработки. Правда, вам придется позаботиться о лицензии на право использования ***Visibroker 4***. При правильной настройке ключей можно сгенерировать необходимые stub-классы и создать на их основе клиентское и серверное CORBA-приложения. Кроме того, в состав проекта можно включить и необходимые пакетные файлы, которые позволят автоматизировать процесс разработки клиентских и серверных приложений.

При использовании простой схемы наследования, рекомендованной в новом стандарте CORBA, и новых способов регистрации и связывания CORBA-объектов, счастливые обладатели редакции Enterprise пользуются фактически той методикой, которую я назвал *CORBA для "бедных"*.

Резюме

В этой главе я не столько описал, сколько перечислил средства поддержки архитектуры CORBA, которые имеются в составе C++Builder. "За кадром" остались многие особенности архитектуры — служба именования, службы обработки событий, активаторы, IIOP и многое другое. Тем, кто всерьез заинтересуется перспективами и методикой реализации этой архитектуры, я рекомендую загрузить с Web-сервера Borland необходимую документацию и тщательно ее проштудировать.

Microsoft Office и приложения C++

Глава

21

Джей Банкс

Интеграция Microsoft Office с программами, разработанными в среде C++Builder, позволяет создавать множество полезных приложений — от простых утилит, помогающих в ежедневной работе, до сложных программ, использующих компоненты Microsoft Office в качестве текстовых процессоров и средств работы с электронными таблицами. В этой главе на множестве примеров будет показано, как организовать совместную работу программ на C++ и компонентов Microsoft Office.

Преимущества интеграции программ на C++ и Microsoft Office

Включение компонентов офисных приложений в прикладные программы позволяет значительно ускорить разработку программного продукта, избавляя программистов прикладных программ от дублирования стандартных средств работы с текстами и электронными таблицами. Я принадлежу к тем программистам, которые всегда не прочь использовать малейшую возможность не делать уже выполненную кем-то работу.

При работе с Microsoft Office задача пользователя значительно облегчается тем, что компоненты этого набора имеют встроенные средства программирования — язык VBA (Visual Basic for Applications), — которые позволяют довольно легко создавать макрокоманды и с их помощью автоматизировать выполнение часто повторяющихся процедур. Например, знакомую каждому процедуру "вырезать–вставить" можно запрограммировать в форме макроса переноса фрагмента текста.

Но использованием макросов в составе компонентов Microsoft Office далеко не исчерпываются возможности адаптации этого продукта к нуждам определенного пользователя (или пользователей определенной категории). Дело в том, что любой компонент Microsoft Office располагает всеми характерными чертами сервера автоматизации, а это открывает широкие перспективы перед программистом, работающим в среде C++Builder и знакомым с методикой использования средств автоматизации ActiveX.

Из всего этого следует, что если в разрабатываемом приложении необходимо выполнять редактирование и обработку текстовой информации, программисту совершенно не нужно "изобретать велосипед", а следует грамотно организовать обращение к Microsoft Word как к серверу автоматизации и воспользоваться всеми его огромными возможностями в части выполнения подобного рода процедур. Конечно, для этого необходимо, чтобы на компьютере пользователя был установлен пакет программ Microsoft Office. Тогда можно из программы на C++ запустить нужный компонент Microsoft Office (или получить частичный контроль над уже запущенным приложением) и использовать его для выполнения запросов со стороны основной программы. Тем же способом можно оформлять отчеты в стандартном формате и выполнять множество других работ с документами. Может быть, то, что "умеет" Word, можно реализовать, запрограммировав соответствующим образом компонент `TQuickRep`, но документ Microsoft Word можно переслать по электронной почте и при этом использовать все возможные средства соблюдения конфиденциальности переписки, гарантирующие, что документ будет прочтен только тем, кому он предназначен.

Включая компоненты Microsoft Office в разрабатываемое приложение C++Builder, программист получает следующие преимущества.

1. *Доступ к хорошо отработанным программам* и интерфейсам для выполнения рутинных процедур обработки документов.

   Если логика работы приложения требует обработки текстовой информации или данных в форме электронных таблиц, программисту нет нужды разрабатывать собственные средства такой обработки или приобретать их у третьих фирм. Пользователь разработанной программы получает возможность работать с хорошо известными ему средствами (а кто из пользователей ПК сейчас не умеет работать с Word или Excel) и

будет несказанно рад, что для работы с разработанным вами приложением ему не придется менять свои привычки.

2. *Доступ к общепринятым видам и форматам документов*.

   Возможность не возиться в разрабатываемой программе с форматами файлов хранения документов. Вместо того чтобы самостоятельно извлекать из файлов необходимую информацию, можно воспользоваться хорошо отработанными средствами работы с таким файлами, уже имеющимися в компонентах Microsoft Office.

3. *Автоматизация решения поставленной задачи*.

   Если в процессе выполнения программы требуется повторять одни и те же операции с множеством разных документов, то с помощью средств автоматизации Microsoft Office это реализуется практически тривиально.

Интеграция компонентов Microsoft Office в программу на C++

Широкие возможности включения компонентов Microsoft Office в другие приложения обеспечивается в первую очередь тем, что эти компоненты располагают множеством общедоступных интерфейсов, позволяющих использовать их как в качестве серверов автоматизации OLE, так и в качестве COM-объектов. Среда C++Builder предоставляет разработчику два способа обращения к этим серверам — с помощью компонента `TOleContainer` и как к объекту автоматизации.

Выбор определяется спецификой задачи, которую должен решать компонент Microsoft Office в разрабатываемом интегрированном приложении. Компонент `TOleContainer` лучше всего использовать в тех случаях, когда требуется фактически включить тот или иной компонент Microsoft Office в создаваемую программу, т.е. сделать его частью этой программы. Но с другой стороны, когда требуется получить более гибкие возможности управления работой компонента, лучше использовать `OleVariant` и обращаться к компонентам Microsoft Office как к объектам автоматизации.

Использование `TOleContainer`

Включить компонент Microsoft в приложение на C++ проще всего с помощью компонента `TOleContainer`, который имеется в комплекте C++Builder. Создайте новое приложение и перетащите компонент `TOleContainer` на поле экранной формы нового приложения. Настройте его размеры таким образом, чтобы зазор между рамкой компонента и рамкой окна приложения со всех сторон был порядка 50 пикселей.

После этого сделайте двойной щелчок на поле вставленного компонента `OleContainer` — среда C++Builder выведет на экран диалоговое окно Insert Object, которое будет выглядеть примерно так, как на рис. 21.1. В поле списка следует отыскать элемент `Microsoft Word document` и щелкнуть затем на кнопке OK.

После этого сохраните проект, скомпилируйте его и запустите на выполнение. На экране появится экранная форма приложения, в центре которой будет пустое поле. Сделайте на нем двойной щелчок кнопкой мыши. После некоторого “размышления” программа трансформирует окно приложения, и появится до боли знакомое окно Microsoft Word, в котором, правда, не будет открыт никакой документ. Когда закончите эксперименты с этим окном, закройте его и вернитесь в среду C++Builder.

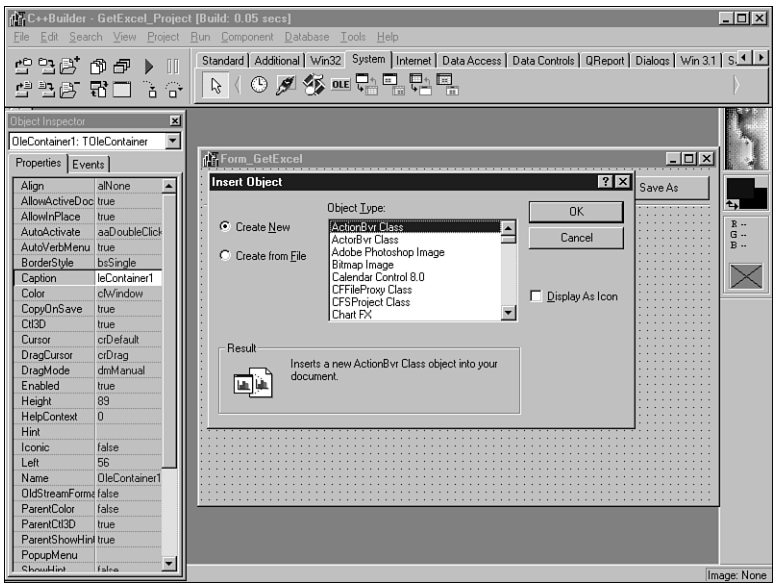


Рис. 21.1. Диалоговое окно Insert Object

Щелкните на поле компонента `OleContainer` правой кнопкой мыши и выберите в контекстном меню команду Delete Object. Тем самым вы отсоедините приложение от Microsoft Word. Добавьте в экранную форму три кнопки, назовите их **`Button_Word`**, **`Button_Excel`** и **`Button_PPoint`**, а в качестве надписей на кнопках установите, соответственно, **`Word`**, **`Excel`** и **`PowerPoint`**. Кнопки разместите ниже поля компонента `OleContainer`.

Добавьте в модуль экранной формы обработчики событий `OnClick`, тексты которых приведены в листинге 21.1. В папке `TOleContainer` на прилагаемом к книге компакт-диске находится проект `TOleContainer_Demo.bpr`, в файлах которого вы найдете полный программный код этого приложения.

Листинг 21.1. Присоединение объектов Word, Excel и PowerPoint

```
//-- Кнопка "Word"
void __fastcall TForm_OleContainer::Button_WordClick(
                TObject *Sender)
{
    OleContainer1->CreateObject("Word.Document",false);
    OleContainer1->DoVerb(ovShow);
}

//-- Кнопка "Excel"
void __fastcall TForm_OleContainer::Button_ExcelClick(
                TObject *Sender)
{
    OleContainer1->CreateObject("Excel.sheet",false);
    OleContainer1->DoVerb(ovShow);
}
```

```
//-- Кнопка "PowerPoint"
void __fastcall TForm_OleContainer::Button_PPointClick(
                TObject *Sender)
{
    OleContainer1->CreateObject("Powerpoint.show",false);
    OleContainer1->DoVerb (ovShow);
}
```

Опять скомпилируйте программу и запустите ее на выполнение. Если в окне работающей программы вы сделаете двойной щелчок на пустом поле контейнера, будет сгенерирована исключительная ситуация и появится сообщение об ошибке `Operation not allowed on an empty OLE container` (Операция не выполнена, поскольку контейнер OLE пуст). Щелкните на кнопке OK и продолжите работу с приложением.

Щелкните на одной из трех кнопок ниже поля контейнера. После щелчка на кнопках Excel или Word на месте поля контейнера откроется окно документа Excel или Word, соответственно, а вместо главного меню экранной формы будет выведено главное меню Excel или Word (рис. 21.2).

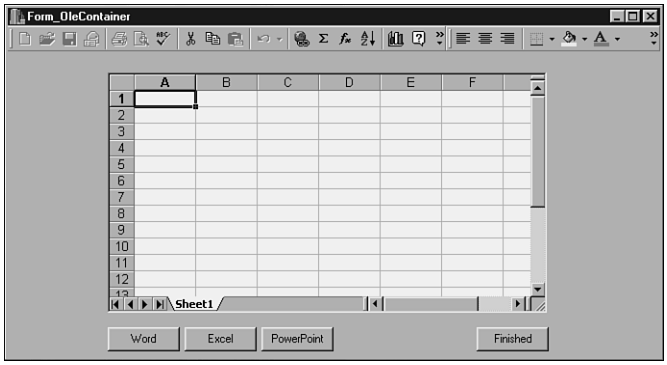


Рис. 21.2. В программе `OleContainer` вызван компонент Excel из состава Microsoft Office

После щелчка на кнопке PowerPoint поле контейнера изменится, но меню приложения останется прежним. Происходит это по той причине, что компонент PowerPoint реализован в Microsoft Office не так, как компоненты Word и Excel. Об этом не следует забывать при создании интегрированных приложений.

Использование механизмов автоматизации

Использование механизмов автоматизации открывает более широкие возможности при интеграции компонентов Microsoft Office в приложение C++Builder, чем использование класса `TOleContainer`. Разработчик получает возможность программно контролировать свойства каждого объекта и его поведение. Естественно, это потребует и определенных усилий при программировании.

В структуре каждого объекта автоматизации имеется множество свойств и методов, доступных программисту в процессе работы с этим объектом. Методика работы с этими свойствами и методами аналогична методике работы со свойствами и методами элементов управления в экранной форме.

Среда C++Builder поддерживает два способа работы с объектами автоматизации — через библиотеку типов и через механизм автоматизации OLE. В файле библиотеки типов содержатся описания всех интерфейсов приложения (в данном случае компонента Microsoft Office), типов, структур и классов, которые используются для работы с этим приложением. Использование библиотеки типов имеет то преимущество, что при этом выполняется проверка типов, но за это приходится кое-чем заплатить — либо самостоятельно сформировать новый файл библиотеки типов с помощью меню Import Type Library, либо получить такой файл от разработчика включаемого приложения. Компиляция программы, в которую через библиотеку типов включен компонент, происходит несколько медленнее, чем обычной программы.

Механизм автоматизации OLE предполагает, что разработчику известны имена интерфейсов, свойств и методов объекта (объектов) автоматизации. Но при использовании этого механизма не выполняется проверка типов — предполагается, что в распоряжении разработчика имеется полная документация на объект автоматизации и он самостоятельно организует приведение типов. Компиляция программы, в которой используется механизм автоматизации OLE, выполняется гораздо быстрее, чем в случае использования файла библиотеки типов.

В большинстве рассматриваемых в этой главе примеров мы будем использовать механизм автоматизации OLE, но рассмотрим и те средства, которыми располагает среда C++Builder для работы с файлами библиотек типов. Как уже отмечалось, использование механизма автоматизации OLE возможно только в том случае, когда имеется полная документация на объекты автоматизации. К счастью, компоненты Microsoft Office поступают с полным комплектом необходимой документации (в электронном виде).

Запустите Word, а затем — редактор Visual Basic (нажмите <Alt+F11> или вызовите команду меню Tools⇨Macro⇨Visual Basic Editor (Сервис⇨Макрос⇨Редактор Visual Basic)). В результате на экране откроется окно редактора Visual Basic, в котором можно просмотреть файлы справки, касающиеся интерфейса. Если это окно автоматически не откроется, введите в поле поиска предметного указателя справки слово `Application` и найдите описание объекта `Application` программы Word. Описание на экране должно выглядеть примерно так, как на рис. 21.3.

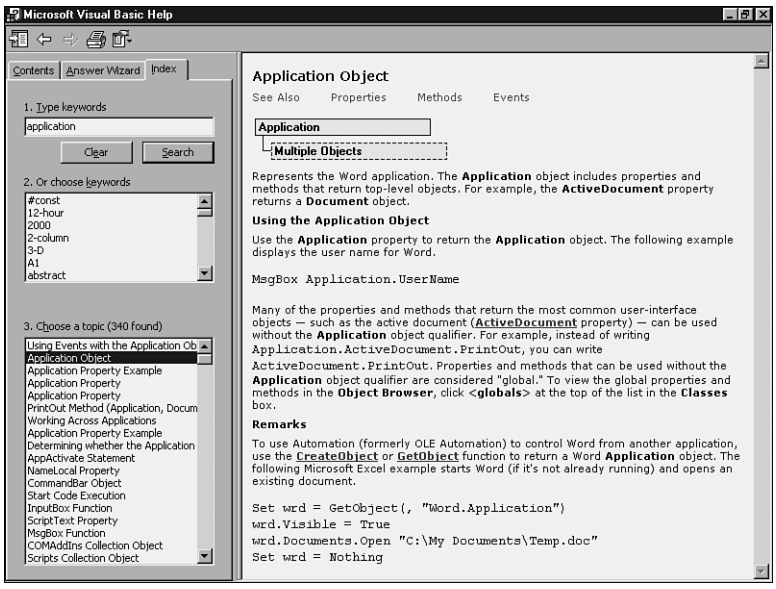


Рис. 21.3. Фрагмент файла справки Microsoft Office

В этом файле справки можно получить информацию о свойствах, методах и событиях объекта, который вас интересует, а также другие полезные сведения. Следует отметить, что, хотя вы открываете этот файл справки в приложении Word, имеющаяся в нем информация касается всех компонентов Microsoft Office.

Использование объектов автоматизации и переменных типа `Variant`

В среде C++Builder переменные типа `Variant` значительно упрощают работу с объектами автоматизации. Такие переменные могут содержать объекты автоматизации, а функции, обслуживающие эти объекты автоматизации, возвращают значения типа `Variant`.

Ниже перечислены методы получения доступа и манипулирования объектами автоматизации:

- `CreateObject()`;
- `GetActiveObject()`;
- `OleFunction()`;
- `OleProcedure()`;
- `OlePropertyGet()`;
- `OlePropertySet()`.

Методы `CreateObject()` и `GetActiveObject()` используются, соответственно, для формирования и получения ссылки на активный объект автоматизации. Назначение остальных методов понятно из их названий. Нужно только отметить, что методы `OleFunction()`, `OleProcedure()`, `OlePropertyGet()` и `OlePropertySet()` являются по сути оболочками метода `Exec()`.

Метод `Exec()` используется C++Builder для запуска на выполнение процедуры или функции, а также для извлечения и установки значений свойств. При выполнении этих операций метод обращается к объектам четырех специальных классов:

- `Function`;
- `Procedure`;
- `PropertyGet`;
- `PropertySet`.

Класс `Function` используется для определения функций, класс `PropertyGet` — для извлечения значения свойства и т.д. Объекты этих классов должны быть определены заранее — до того как они будут использоваться методом `Exec()`. После завершения работы с объектами они очищаются с помощью метода `ClearArgs()` перед повторным использованием. Рассмотрим кратко на примере, как это делается.

Создайте новое приложение, разместите в его экранной форме кнопку `Button_LaunchWord` с надписью **`Launch Word`**. В обработчик события `OnClick` этой кнопки добавьте программный код из листинга 21.2. Этот пример вы сможете найти и на прилагаемом компакт-диске, в папке `AutomationObj` (наименование файла проекта — `automation_objects.bpr`).

Листинг 21.2. Запуск Word

```
void __fastcall TForm_Automation::Button_LaunchWordClick(
               TObject *Sender)
{
    Variant word_app;
    Variant word_docs;
    Variant word_this_document;
    Variant word_range;
```

```
    word_app = Variant::CreateObject("word.application");
    word_docs = word_app.OlePropertyGet("documents");

    word_docs.OleProcedure("Add");

    word_app.OlePropertySet("Visible", (Variant)true);
}
```

Обработчик с теми же функциональными возможностями, но с использованием метода `Exec()` показан в листинге 21.3. Добавьте в экранную форму еще одну кнопку, `Button_LaunchWordExec`, с надписью **`Launch with Exec`** и добавьте в ее обработчик события `OnClick` программный код из листинга 21.3.

Листинг 21.3. Запуск Word с помощью метода `Exec()`

```
void __fastcall TForm_Automation::Button_LaunchWordExecClick(TObject *Sender)
{
    Variant word_app;
    Variant word_docs;
    Variant word_this_document;
    Variant word_range;

   //-- Определение функции Documents
    Function Documents("Documents");
   //-- Определение функции добавления документов
    Function AddDocument("Add");
   //-- Определение свойства видимости
    PropertySet Visibility("Visible");

    word_app = Variant::CreateObject("word.application");
    word_docs = word_app.Exec(Documents);

    word_docs.Exec(AddDocument);

    Visibility << True;
    word_app.Exec(Visibility);
}
```

На рис. 21.4 показано окно программы `LaunchWord` во время выполнения.

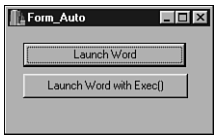


Рис. 21.4. Окно программы `LaunchWord` во время выполнения

Единственное, что осталось сделать в этом примере, — обеспечить доступ компилятору к файлу библиотеки `utilcls.h`. Использование переменных типа `Variant` при работе с механизмом автоматизации OLE требует знания структуры класса `TAutoArgs`, который определен в файле `utilcls.h`. Поэтому включите в файл реализации директиву включения:

```
#include <utilcls.h>
```

Скомпилируйте программу и запустите ее на выполнение. Когда на экране появится окно программы `LaunchWord`, щелкните на кнопке Launch Word — будет запущено новое приложение Word с пустым документом, причем новый экземпляр приложения будет создан и в том случае, если приложение Word уже работает.

Защита от макровирусов

Одно из достоинств использования механизма автоматизации состоит в том, что программист получает в свое распоряжение множество средств управления включенным в приложение "посторонним" компонентом. Но эта медаль имеет и обратную сторону — программист берет на себя полную ответственность за правильность настройки подключенного компонента. Если с помощью этого компонента открывается документ, содержащий макровирус, то офисное приложение, даже обнаружив "заразу", полагает, что программист отвечает за свои действия, и не препятствует выполнению запрограммированной процедуры.

Если в приложении вы собираетесь автоматизировать открытие документа, то необходимо самостоятельно предпринять меры, не позволяющие макровирусу сделать свое "грязное" дело. Убедитесь, что на компьютере установлена достаточно свежая антивирусная программа, которая защитит разработанное приложение от открытия такого документа, а следовательно, избавит вас от возможных неприятностей.

Использование языка Word Basic

Когда фирма Microsoft выпустила "в свет" версию Word 97, в ней была предусмотрена возможность создания несложных программ на языке Word Basic. Хотя в следующей версии, Word 98, был оставлен только язык Visual Basic for Applications, Microsoft сочла необходимым оставить в этой версии и прежний интерфейс с Word Basic.

Доступ к этому интерфейс открывается через объект автоматизации `word.basic`, и, хотя он и не предоставляет таких же широких возможностей, как VBA, существуют ситуации, когда имеет смысл его использовать. Например, это удобно в приложениях, которые предназначены для работы с разными версиями компонентов Microsoft Office, т.е. рассчитаны на самый широкий круг пользователей.

Если вы предполагаете, что среди потенциальных потребителей есть те, которые пользуются Office 97, то лучше ориентироваться на Word Basic, а не на VBA. Те, кто пользуется более современными версиями Microsoft Office, смогут прекрасно работать с разработанными вами приложениями, но кроме них среди пользователей будут и те, кто еще не успел обновить версию, а значит, у вас будет более широкий круг пользователей.

Особенности интеграции Word

Microsoft Office представляет собой довольно сложную программную систему, которая позволяет независимым разработчикам обращаться к компонентам Office через некоторые из внутренних интерфейсов этого комплекта. В этой разделе будет представлен обзор этих интерфейсов и способы их использования при выполнении часто встречающихся процедур загрузки, сохранения и создания документов Word.

Коллекции

Коллекция используется для хранения группы объектов Word. Каждая коллекция имеет два ключевых ресурса, которые и позволяют "добраться" к хранящимся в ней объектам, — свойство `count` и метод `item()`. Свойство `count` несет информацию о количестве объектов в коллекции, а метод `item()` используется для получения ссылки на выбранный объект в коллекции либо по номеру, либо по имени (если, конечно, объект именованный). Некоторые коллекции имеют другие свойства и методы, которые открывают дополнительные возможности для работы с ними. Прекрасным примером тому может служить коллекция `documents`, которая располагает методом `add()`, формирующим новый документ.

Объект приложения

Объект приложения является нервным центром Microsoft Word и отправной точкой для любой программы, которая нуждается в услугах Word. После того как объект приложения создан или получена ссылка на него, можно получить ссылки и на коллекции, и свойства, необходимые для работы с Word.

При работе с Word существует два способа получения объекта приложения — нужно либо создать новый объект, либо получить ссылку на существующий. Объекты типа `Variant` предоставляют в распоряжение программиста два метода, позволяющих выполнить такие процедуры: `CreateObject()` и `GetActiveObject()`. Метод `CreateObject()` используется для создания нового экземпляра объекта автоматизации, а при вызове метода `GetActiveObject()` предпринимается попытка отыскать в операционной системе созданный ранее объект и получить на него ссылку. Если попытка создания или поиска при выполнении любого из этих методов не удалась, генерируется исключительная ситуация типа `EoleSysError`.

Для получения ссылки на объект автоматизации Word используется идентификатор программы `word.application`. При необходимости работать с объектом приложения попробуйте сначала получить ссылку на работающий объект. Это позволит избежать параллельного существования в операционной системе нескольких однотипных объектов и избавит конечного пользователя от необходимости ждать, пока объект будет сформирован.

Создайте новое приложение и включите в его экранную форму кнопку `Button_LaunchWord`, на которой будет "красоваться" надпись Launch Word. В обработчик события `OnClick` этой кнопки добавьте программный код из листинга 21.4.

Полный программный код этого примера вы найдете в папке `LaunchWord` на прилагаемом к книге компакт-диске (имя файла проекта — `example3.bpr`).

Листинг 21.4. Избирательный запуск Word

```
void __fastcall TForm1::Button_LaunchWordClick(TObject *Sender)
{
try
    {
        my_word = Variant::GetActiveObject("word.application");
    }
    catch (...)
    {
        //-- Не удается отыскать активный объект Word.
        //-- Сформировать новый объект!
        try
        {
            my_word = Variant::CreateObject("word.application");
        }
        catch (...)
        {
            //-- Не удается сформировать новый объект Word.
            Application->MessageBox(
                       "Unable to obtain Word automation object",
                       "Error:", MB_OK|MB_ICONERROR);
        }
    }
    my_word.OlePropertySet("Visible", (Variant)True);
}
```

Добавьте в класс экранной формы приложения новый член — объект типа Variant:

```
Variant  my_word;
```

Скомпилируйте программу и запустите ее на выполнение. После того как на экране появится окно программы, щелкните на кнопке Launch Word. Если в текущий момент на компьютере уже работала программы Word, никаких видимых изменений после эго не произойдет (в нашей программе не запрограммированы операции с этим объектом автоматизации), а если Word ранее не был запущен, то будет сгенерирована исключительная ситуация и C++Builder прекратит работу. Этого не случится, если программа будет запущена автономно, а не из среды C++Builder.

Работа с документами

Получив доступ к объекту приложения, можно обратиться к его коллекции documents. Коллекция представляет собой объект с собственной структурой, т.е. обладающий собственными свойствами и методами. В число методов объекта коллекции documents входят и такие, которые позволяют получить список уже открытых документов, открыть какой-либо из существующих документов или создать новый.

Ссылка на коллекцию documents может быть получена при обращении к объекту приложения:

```
my_documents = my_word.OlePropertyGet("Documents");
```

Мы добавим этот оператор в программу чуть позже.

Создание нового документа

Метод Add используется для создания нового документа Word. Этот метод имеет несколько необязательных аргументов, которые можно использовать для определения свойств создаваемого документа.

Для создания чистого нового документа достаточно вызвать метод Add() без передачи ему каких-либо аргументов. Если документ планируется создать на основе шаблона, вызовите метод, передав ему в качестве аргумента имя шаблона. Добавьте в приложение новую кнопку Button_CreateDocument с надписью **Create Document**. В обработчик события OnClick этой кнопки включите программный код из листинга 21.5.

Листинг 21.5. Создание нового документа Word

```
void __fastcall TForm1::Button_CreateDocumentClick(
               TObject *Sender)
{
    Variant  this_doc;
    long  doc_count;


    my_docs  = my_word.OlePropertyGet("Documents");
    this_doc  = my_docs.OleFunction("Add");

    doc_count  = my_docs.OlePropertyGet("Count");
    ShowMessage((AnsiString)"there are " + doc_count +
                           " document(s) ");
    // "Имеется " + doc_count + " документов");
}
```

В определение класса экранной внесите новый член:

```
Variant  my_docs;
```

Скомпилируйте программу и запустите ее на выполнение. Результат должен выглядеть, как на рис. 21.5. Щелкните на кнопке Create Document — это должно привести к генерированию исключительной ситуации и появлению на экране сообщения `Variant does not reference an automation object` (Переменная типа `Variant` не ссылается на объект автоматизации). Прежде чем пытаться сделать что-нибудь с помощью программы Word, нужно убедиться, что получена ссылка именно на этот объект автоматизации. В окне сообщения щелкните на кнопке Continue и после этого — на кнопке Launch Word в окне программы. Вот теперь можно щелкнуть на кнопке Create Document. В Word будет добавлен новый пустой документ, а на экране появится окно сообщения, в котором будет выведена информация о текущем количестве открытых документов.

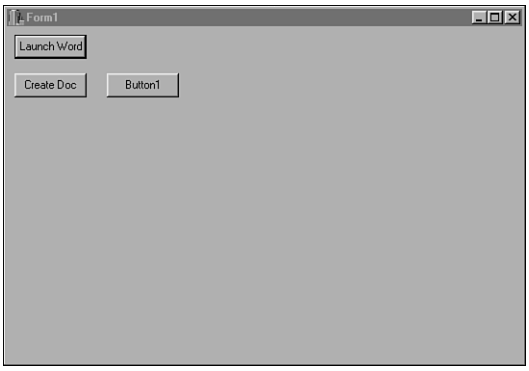


Рис. 21.5. Окно программы `example3`

Использование метода `Save` для сохранения документов

Освоив методику создания нового документа, посмотрим, каким образом можно его сохранить. Это можно сделать с помощью одного из методов сохранения или с помощью метода `SaveAs` самого объекта документа.

Существует три способа сохранения: сохранить отдельный документ, сохранить все открытые документы и сохранить версию документа. Все зависит от того, какой объект сохраняется. Объекты `document` или `version` сохраняют собственные копии, а объект коллекции `documents` сохраняет все открытые документы.

Для того чтобы продемонстрировать, как пользоваться методами сохранения, добавим в экранную форму программы еще одну кнопку, `Button_QuickSave`, с надписью **`Quick Save`**. В обработчик события `OnClick` этой кнопки включите программный код из листинга 21.6.

Листинг 21.6. Сохранение документа Word

```
void __fastcall Tform1::Button_QuickSaveClick(TObject *Sender)
{
    Variant  this_doc;

    this_doc = my_word.OlePropertyGet("ActiveDocument");
    this_doc.OleProcedure("Save");
}
```

Скомпилируйте программу и запустите ее на выполнение. Щелкнув на кнопке Launch Word, запустите Word, а затем щелкните на кнопке Create Document, чтобы создать новый документ. После этого щелкните на кнопке Quick Save. Приложение Word выведет диалоговое окно Save As, поскольку при вызове метода ему не было передано имя файла. Закройте программу и вернитесь в среду C++Builder.

Добавьте в экранную форму еще одну кнопку, `Button_QuickSaveAs`, с надписью **`Quick Save As`**, а в обработчик события `OnClick` этой кнопки вставьте программный код из листинга 21.7.

Листинг 21.7. Сохранение документа Word с новым именем

```
void __fastcall TForm1::Button_QuickSaveAsClick(
                TObject *Sender)
{
    Variant  this_doc;

    this_doc  = my_word.OlePropertyGet("ActiveDocument");
    this_doc.OleProcedure("SaveAs",(Variant)"c:\ \ saved_as. doc");
}
```

Скомпилируйте программу и запустите ее на выполнение. После запуска Word и создания нового документа щелкните на кнопке Quick Save As. На этот раз Word сохранит документ без лишних вопросов, поскольку имя файла передано аргументом вызова метода `SaveAs`.

Метод `SaveAs` принимает в качестве аргумента не только имя файла, поэтому при вызове этого метода можно передавать и другую информацию. Желательно при этом непосредственно использовать метод `Exec()`, а не пользоваться его оболочкой `OleProcedure()`.

Открытие существующего документа

Выше было описано программирование создания и сохранения документов, а теперь пришла очередь операции открытия ранее созданного документа. Ее можно выполнить с помощью метода `Open` объекта коллекции `documents`.

Если имя документа известно, его можно передать в качестве аргумента вызова метода `Open`. Добавьте в экранную форму нашей программы кнопку `Button_QuickOpen` и "закажите" для нее надпись **`Quick Open`**. После щелчка на этой кнопке программа должна запустить операцию загрузки документа в Word.

В обработчик события `OnClick` новой кнопки добавьте программный код, представленный в листинге 21.8. Скомпилируйте программу и запустите ее на выполнение. После запуска Word щелкните на кнопке Quick Open — Word загрузит документ, сохраненный в предыдущем сеансе работы с программой.

Листинг 21.8. Загрузка документа в Word

```
void __fastcall Tform1::Button_QuickOpenClick(TObject *Sender)
{
    Variant    this_doc;
    my_docs = my_word.OlePropertyGet("Documents");

    this_doc = my_docs.OleFunction("Open",
                    (Variant)"c:\ \ saved_as.doc");
}
```

Извлечение текста из документа Word

До сих пор мы рассматривали базовые операции с файлами документов Word: создание, сохранение и открытие документов. В этом разделе посмотрим, как извлечь из документа Word полезную информацию. В процессе работы будет создано новое приложение, которое сможет открывать документ, извлекать из него список слов, сортировать его и создавать из этого списка новый документ.

Ключевыми элементами процедуры извлечения слов являются объект `range` и коллекция `words`. Объект `range` задает фрагмент документа, который будет извлекаться и обрабатываться в программе. В данном приложении мы будем извлекать все содержимое документа и сформировать коллекцию слов, содержащихся в нем.

Объект `range` формируется на основании информации о положении первого и последнего символов извлекаемого фрагмента. Если не задавать этих параметров, извлекается все содержимое документа. При извлечении фрагмента можно явно задавать позиции начального и конечного символов либо воспользоваться тем, что многие объекты располагают собственными объектами типа `range`. В листинге 21.9 представлен фрагмент программного кода, в котором демонстрируются разные способы заполнения объекта `range`.

Листинг 21.9. Выбор фрагмента через `Range` и через `Paragraph`

```
//-- Извлечение документа целиком.
my_range = my_docs.OleFunction("Range");

//-- Извлечение первых 33 символов.
my_range = my_docs.OleFunction("Range", (Variant) 0, Variant(33));

//-- Извлечение второго абзаца.
my_paras = my_docs.OlePropertyGet("Paragraphs")
my_para  = my_paras.OleFunction("Item",(Variant) 2);
p_range  = my_para.OlePropertyGet("Range");
my_range = my_docs.OleFunction("Range", p_range.OlePropertyGet("Start"),
                                        p_range.OlePropertyGet("End"));
```

После извлечения определенного фрагмента нужно сформировать из его содержимого коллекцию `words`. Это делается очень просто с помощью свойства `words` объекта `range`. Сформировав коллекцию `words`, можно получить список слов, содержащихся в документе.

Создайте новое приложение, в экранную форму которого поместите элемент управления типа *список* (оставьте предлагаемое по умолчание наименование этого элемента) и кнопку `Button_LoadWords`, которая должна иметь надпись **`Load Words`**. В обработчик события `OnClick` этой кнопки включите программный код из листинга 21.10.

Файлы программного кода этого примера имеются на прилагаемом компакт-диске (папка `LoadWords`, файл проекта `example_4.bpr`).

Листинг 21.10. Обработчик щелчка на кнопке в программе `LoadWords`

```
void __fastcall Tform1::Button_LoadWords(TObject *Sender)
{
    Variant  my_doc;
    Variant  my_word;
    long  word_count;
```

```
long  index;

    try
{
    my_word = Variant::GetActiveObject("word.application");
}
catch (...)
{
    //-- Перед запуском этой программы нужно запустить Word.
    //-- Вывести на экран сообщение и прекратить работу.
    ShowMessage("Please open word and load a document and retry");
      // "Запустите Word, загрузите в него документ и повторите
      //  вызов программы.";
    return;
}

    //-- Извлечь содержимое текущего документа Word.
    my_doc    = my_word.OlePropertyGet("ActiveDocument");

    //-- В данном случае объект 'range' не создается.
    //-- Извлекаются все слова из текущего документа.
    my_words  = my_doc.OlePropertyGet("Words");

    //-- Количество слов в документе.
    word_count    = my_words.OlePropertyGet("Count");

    //-- Скопировать все слова в список.
    for (index=1;index<word_count;index++)
    {
        ListBox1->Items->Append(my_words.OleFunction("Item",
                              (Variant)index));

    }
}
```

Скомпилируйте программу и запустите ее на выполнение. Запустите приложение Word и введите в пустой документ несколько слов по своему выбору. После этого вернитесь в окно нашей программы и щелкните на кнопке Load Words. Список в экранной форме должен заполниться словами из открытого документа.

Обратите внимание на то, что, кроме собственно слов, в список попали и знаки пунктуации, и некоторые управляющие символы. Word считает эти элементы словами. Хорошо это или плохо — зависит от цели, которую преследует разработчик программы.

Если документ достаточно большой, то на заполнение списка уйдет довольно много времени. Это — побочный эффект использования описанной методики извлечения слов из документа, поскольку заполнение списка происходит в цикле, по одному слову за цикл.

Но, к счастью, Word содержит все необходимое, чтобы избавить программиста от необходимости извлекать слова из документа по одному. В Word имеется объект `selection`, с помощью которого можно обратиться ко всему тексту документа как к единому объекту и скопировать его в стандартный элемент управления C++Builder в частности в элемент типа `RichEdit` (рис. 21.6).

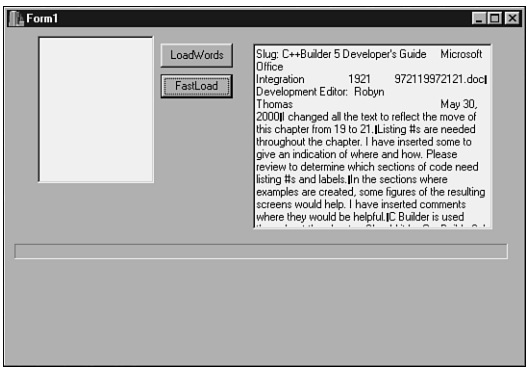


Рис. 21.6. Элемент управления типа `RichEdit`, в который загружен текст из документа Word

Добавьте в экранную форму программы еще одну кнопку (присвойте ей наименование `Button_FastLoad`, а на кнопке должна "красоваться" надпись **`Fast Load`**) и элемент управления `RichEdit`. В обработчик события `OnClick` новой кнопки добавьте программный код, приведенный в листинге 21.11. После повторной компиляции программы и запуска ее на выполнение щелкните на кнопке Fast Load. Буквально через мгновение содержимое документа будет выведено в поле элемента управления `RichEdit`. После того как текст окажется в буфере элемента управления C++Builder, из него будет гораздо проще извлечь отдельные слова.

Листинг 21.11. Обработчик щелчка на кнопке Fast Load

```
void __fastcall TForm1::Button_FastLoadClick(TObject *Sender)
{
    Variant    my_doc;
    Variant    my_word_app;
    Variant    my_words;
    Variant    my_selection;

    try
    {
        my_word_app =
         Variant::GetActiveObject("word.application");
    }
    catch (...)
    {
         //-- Перед запуском этой программы нужно запустить Word.
         //-- Вывести на экран сообщение и прекратить работу.
        ShowMessage(
              "Please open word and load a document and retry");
        // "Запустите Word, загрузите в него документ и повторите
        //  вызов программы.";
        return;
    }

    //-- Извлечь содержимое текущего документа Word.
    my_doc    = my_word_app.OlePropertyGet("ActiveDocument");
```

```
    //-- Выбрать весь текст документа.
    my_doc.OleProcedure("Select");
    my_selection = my_word_app.OlePropertyGet("Selection");

    RichEdit1->Clear();
    RichEdit1->Text = my_selection.OlePropertyGet("Text");
}
```

Вставка объектов в документы Word

Итак, с загрузкой документа Word и извлечением текста из него мы разобрались. Теперь рассмотрим, каким образом можно вставить информацию в такой документ. Создадим новую программу, которая будет формировать новый документ и поместит в него слова из прежнего документа в алфавитном порядке. Для этого нам понадобится объект `range`, о котором уже шла речь выше.

Вставка текста в документ Word

Существуют два метода, пригодных для вставки текста в документ Word: `InsertBefore` и `InsertAfter`. Первый метод вставляет текст в начало, а второй — в конец текущего объекта `range`. При выполнении этих методов диапазон "захвата" объекта `range` соответственно увеличивается. Какой из методов выбрать в конкретном случае, зависит от того, куда нужно вставить новые строки текста.

Вставка объектов в документ Word

Помимо собственно текста, в документ можно вставлять и объекты форматирования — значки абзацев, разрыва страниц и т.п. Другие методы позволяют вставлять в документ таблицы, изображения и прочие объекты такого рода. В данном разделе мы ограничимся вставкой в документ Word таблиц и текстовых фрагментов.

Выше уже было кратко сказано о том, что операции вставки в документ Word "привязаны" к объекту `range`. Примерно по этой же схеме выполняется вставка и прочих типов объектов. Отличие только в том, что иногда при этом используется не объект типа `range`, а коллекция, к которой принадлежит новый объект, и используется метод `Add`, который заменяет содержимое текущего объекта `range`, если только этот объект не находится в "вырожденном" состоянии. Под "вырожденным" понимается такой объект, у которого позиции начального и конечного символов совпадают. Работа с вырожденным объектом `range` позволяет избежать удаления его прежнего содержимого при вставке нового.

Программный код вставки новой таблицы в конец документа Word представлен в листинге 21.12.

Листинг 21.12. Создание и вставка таблицы в документ Word

```
    long    num_rows    = 5;
    long    num_columns = 3;

    //-- Сформировать объект range
    my_doc    = my_word.OlePropertyGet("ActiveDocument");
    my_range  = my_doc.OleFunction("Range");
```

```
//-- Сжатие объекта range
my_range.OleProcedure("Collapse", (Variant)0);

//-- Вставка таблицы
my_tables   = my_range.OlePropertyGet("Tables");
my_table    = my_tables.OleFunction("Add",
                  my_range, (Variant)num_rows,
                  (Variant)num_columns);
```

В этом фрагменте программы сначала формируется объект `range`, который "охватывает" весь документ, а затем объект "сжимается" и переводится в вырожденное состояние, причем конечная позиция остается на месте, а начальная смещается к конечной. Если бы метод `Collapse` был вызван без передачи ему аргументов или с аргументом `1`, то начало диапазона осталось бы на месте при сжатии, а конец переместился к началу.

Далее в программе в коллекцию `tables` добавляется новый объект `table`. Обратите внимание на то, что в методе `Add` объект `range` используется в качестве аргумента. Этот аргумент указывает методу `Add`, куда вставить таблицу.

Программа составления словаря

Я обещал продемонстрировать программу, которая будет извлекать слова из документа Word и формировать новый документ, включающий отсортированный список извлеченных слов. Эту программу составления словаря мы и рассмотрим в данном разделе. Диалоговое окно этой программы представлено на рис. 21.7. Набор файлов исходных программных кодов этой программы вы найдете на прилагаемом к книге компакт-диске (папка `VocabA`, файл проекта `vocab_proj.bpr`).

Рис. 21.7. Диалоговое окно программы составления словаря

Программа загружает документ Word, копирует его содержимое в переменную типа `AnsiString`, а затем извлекает из текста отдельные слова, отыскивая символы пробелов и другие символы-ограничители. Слова помещаются в буфер элемента управления типа *список*, сортируются, дубликаты отбрасываются, а затем отсортированный список передается в новый документ Word в виде таблицы. Именно эта последняя часть программы и представляет для нас особый интерес — она представлена в листинге 21.13.

Одна из функций этой подпрограммы — форматирование таблицы таким образом, что в заголовок колонки выносится имя файла документа, из которого извлечены слова. Это выполняется с помощью коллекции ячеек `cells`, которая входит в таблицу. Для доступа к каждой ячейке таблицы используется номер строки и столбца (отсчет начинаются с `1`, а не с `0`). В структуру объекта `cell` включен метод `merge`, который выполняет слияние содержимого всех ячеек, начиная от "своей" ячейки до ячейки, указанной аргументом. В данном случае (листинг 21.13) объединяется содержимое четырех ячеек и таким образом формируется заголовок таблицы.

Каждая ячейка имеет собственный объект `range`, который играет по отношению к содержимому ячейки ту же роль, что и объект `range` документа по отношению ко всему документу. В этой программе организована рекурсивная процедура вставки в таблицу. Верхняя левая ячейка каждой таблицы содержит новую таблицу 2×2. При вставке текста в эту новую таблицу необходимо получить ссылку на объект таблицы, а затем извлечь ее объекты `range` ее ячеек.

Листинг 21.13. Вставка слов в таблицу

```
void __fastcall TVocabForm::ReportWordsFromList(Variant doc,
                                      TListBox *lb)
{
    Variant    my_tables;
    Variant    this_range;
    Variant    this_table;
    Variant    start_cell;
    Variant    end_cell;
    Variant    cell_range;

    long    num_columns = 4;
    long    num_rows = (lb->Items->Count + num_columns - 1) /
                                            num_columns;
    long    words;
    long    this_column;
    long    this_row;

   //-- Процедура InsertAfter для метода Exec
    Procedure InsertAfter("InsertAfter");

    //-- Создать объект range.
    this_range = doc.OleFunction("Range");

//-- Извлечь коллекцию tables из объекта range
    my_tables = this_range.OlePropertyGet("Tables");

    //-- Создание новой таблицы, которая будет содержать
    //-- все слова и строку заголовка.
    this_table    = my_tables.OleFunction("Add", this_range,
                                      (Variant)(num_rows+1),
                                         (Variant)(num_columns));
    start_cell  = this_table.OleFunction("Cell",
                                   (Variant)1, (Variant)1);
    end_cell    = this_table.OleFunction("Cell", (Variant)1,
                                   (Variant)num_columns);
    start_cell.OleProcedure("Merge", end_cell);

    //-- Сформировать заголовок таблицы.
    cell_range    = start_cell.OlePropertyGet("Range");

    cell_range.Exec(InsertAfter << "Words from ‘" +
                          OpenDialog1->FileName + "’");

    this_column    = 1;
    this_row    = 2;

    //-- Скопировать слова в таблицу...
    for (words=0; words<lb->Items->Count; words++)
    {
```

```
        start_cell    = this_table.OleFunction("Cell",
                                    (Variant)this_row,
                                    (Variant)this_column);
        cell_range    = start_cell.OlePropertyGet("Range");

        InsertAfter.ClearArgs();
        cell_range.Exec(
             InsertAfter << lb->Items->Strings[words]);

        this_column++;
        if (this_column>num_columns)
        {
            this_row++;
            this_column=1;
        }
    }
}
```

Интеграция Excel в приложение C++Builder

В этом разделе мы рассмотрим возможность использования в приложении, созданном в C++Builder, другого компонента Microsoft Office — программы Excel. Как и Word, программа Excel предоставляет большинство своих внутренних механизмов в виде объектов автоматизации и позволяет использовать коллекции для хранения фрагментов содержимого документа.

По собственному опыту вы знаете, что методы работы с Word и Excel очень отличаются, но те принципы доступа к содержимому документа, которые мы использовали при интеграции Word в приложение C++Builder, могут быть успешно использованы и при интеграции Excel.

Формирование ссылки на объект приложения

Сформировать в программе C++ ссылку на объект приложения Excel можно тем же способом, что и ссылку на объект приложения Word. Единственная разница — имя запрашиваемого объекта.

При работе с Excel нам необходимо получить объект `excel.application`, а не `word.application`. Эта процедура выполняется следующим оператором:

```
Variant my_excel = Variant::CreateObject("excel.application");
```

Операции с рабочей книгой Excel

При работе с Word наиболее объемной коллекцией является *документ Word*, который содержит текст, таблицы, изображения и т.п. В Excel дело обстоит несколько по-другому — можно обращаться к *рабочим книгам* (*workbooks*), которые, в свою очередь, состоят из множества *рабочих листов* (*worksheets*). Рабочие листы включают текст, формулы и прочие составляющие документа Excel.

Доступ к рабочим книгам можно получить двумя способами: запросить коллекцию `workbooks` объекта приложения или обратиться к его свойству `activeworkbook`. Объект кол-

лекции позволит воспользоваться всем спектром функций — добавление, открытие, закрытие и т.п., а свойство activeworkbook позволит получить только доступ к текущей рабочей книге.

Создание новой рабочей книги

Новая рабочая книга, как и любой новый объект коллекции, создается с помощью метода Add этой коллекции, в данном случае — коллекции workbooks. Можно воспользоваться шаблоном рабочей книги по умолчанию или каким-либо специализированным шаблоном — его наименование передается в качестве аргумента вызова метода Add. При создании новой рабочей книги программа Excel присваивает ей "сиротское" имя (как подкидышам в приюте) — что-то вроде Book 1, Book 2 и т.д. Это имя затем может быть использовано при сохранении документа.

Во вновь созданную рабочую книгу автоматически включаются "чистые" рабочие листы, количество которых устанавливается текущим значением свойства SheetsInNewWorkbook объекта приложения. Программа формирования новой рабочей книги представлена в листинге 21.14. Все файлы программного кода этого примера вы найдете на прилагаемом к книге компакт-диске (папка GetExcel, файл проекта GetExcel_Project.bpr).

Листинг 21.14. Создание рабочей книги Excel

```
void __fastcall TForm_GetExcel::Button_NewWorkbookClick(
                            TObject *Sender)
{
    Variant    all_workbooks;
    Variant    my_workbook;

    //-- Извлечь объект коллекции workbooks.
    all_workbooks    = my_excel.OlePropertyGet("Workbooks");

    //-- Установить количество рабочих листов равным 1.
 my_excel.OlePropertySet("SheetsInNewWorkbook",(Variant)1);

    //-- Создать новую рабочую книгу.
    my_workbook        = all_workbooks.OleFunction("Add");
}
```

Сохранение рабочих книг

Для сохранения рабочих книг существует несколько способов, но чаще всего используются методы Save и SaveAs. Методы эти работают практически так же, как и их "собратья" в объекте приложения Word. Некоторое отличие вносит тот факт, что при создании рабочей книги ей сразу присваивается имя, а потому при сохранении можно обойтись без запроса имени файла у пользователя. Обращение к методу Save демонстрируется в листинге 21.15.

Листинг 21.15. Сохранение рабочей книги Excel

```
void __fastcall TForm_GetExcel::Button_SaveClick(
                                          TObject *Sender)
{
    Variant  my_workbook =
                  my_excel.OlePropertyGet("ActiveWorkbook");
```

```
    my_workbook.OleProcedure ("Save");
}
```

Метод SaveAs, если не передавать ему аргументов, выполняет те же операции, что и метод Save. Но при передаче методу SaveAs аргумента — имени файла и пути к нему — документ будет сохранен под именем, отличным от имени самого объекта рабочей книги. В листинге 21.16 представлена функция, сохраняющая рабочую книгу с помощью метода SaveAs.

Листинг 21.16. Сохранение рабочей книги Excel с помощью метода SaveAs

```
void __fastcall TForm_GetExcel::Button_SaveAsClick(
                                          TObject *Sender)
{
    Procedure SaveAs("SaveAs");
    Variant  my_workbook =
                 my_excel.OlePropertyGet("ActiveWorkbook");
    my_workbook.Exec(SaveAs << "C:\ \ my_file.xls");
}
```

Открытие рабочей книги

Рабочая книга Excel открывается по той же схеме, что и документ Word, — с помощью метода open коллекции workbooks. Если имя документа известно заранее, оно передается методу open в качестве аргумента (листинг 21.17).

Листинг 21.17. Открытие рабочей книги Excel

```
void __fastcall TForm_GetExcel::Button_OpenClick(TObject *Sender)
{
    Variant all_workbooks = my_excel.OlePropertyGet("workbooks");
    Procedure Open("Open");

    if (OpenDialog1->Execute())
        all_workbooks.Exec(Open << OpenDialog1->FileName);
}
```

Как и в случае обращения к программе Word, метод open можно использовать в качестве средства импортирования файлов в Excel, поскольку он позволяет работать с любыми типами файлов, воспринимаемыми программой Excel. Если планируется работать с большим количеством текстовых файлов, следует отдать предпочтение методу OpenText. Этот метод загружает текст в программу Excel и пытается вставить его в ячейки. Как и метод Open, метод OpenText имеет множество аргументов, позволяющих настроить процесс загрузки текста. Более подробную информацию вы можете получить в файлах справки.

Использование активной рабочей книги

Объект приложения отслеживает, какая из открытых в Excel рабочих книг является текущей. Доступ к текущей рабочей книге осуществляется через свойство activeworkbook объекта приложения. Получив доступ, нужно выбрать один из рабочих листов — объект worksheet из коллекции worksheets — или воспользоваться свойством activesheet для извлечения те-

кущего рабочего листа. Если выбран рабочий лист, который в данный момент неактивен (не является текущим), следует, вызвав метод `activate` этого объекта, сделать его активным.

Затем можно приступать в извлечению информации из этого рабочего листа. Как и в случае Word, информация извлекается при помощи объекта `range` и свойства `range`. Объект `range` задает набор (коллекцию) ячеек или отдельную ячейку. Использование свойства `range` и объекта типа `range` в программе демонстрируется в листинге 21.18.

На заметку

Работая с приложениями, разработанными в Microsoft, нужно очень внимательно относиться к именам объектов и их свойств. Зачастую свойство одного объекта имеет такое же наименование, как и тип другого объекта. Например, доступ к объекту `range` организуется с помощью свойства `range`. Это не такая уж сложная проблема — просто нужно внимательно следить за тем, что вы используете в программе — свойство или объект.

Листинг 21.18. Использование свойства `range`

```
//-- Извлечение ссылки на все ячейки рабочего листа.
my_range = my_worksheet.OlePropertyGet("Range");

//-- Извлечение ссылки на ячейки от A1 до E7.
my_range = my_worksheet.OlePropertyGet("Range", (Variant)"A1:E7");

//-- Извлечение ссылки на отдельную ячейку.
my_range = my_worksheet.OlePropertyGet ("Range", (Variant)"C4");
```

Вставка ячеек в рабочие листы Excel

В структуре каждого объекта типа `range` имеются свойства, которые позволяют вставлять в документ Excel данные и формулы. К таким свойствам, в частности, относятся `value` и `formula`. Действие этих свойств может распространяться в равной степени на несколько ячеек рабочего листа. Проанализируйте представленный в листинге 21.19 обработчик события `OnClick` из программы `GetExcel` (окно этой программы показано на рис. 21.8).

Листинг 21.19. Заполнение ячейки рабочего листа Excel

```
void __fastcall TForm_GetExcel::Button_WorkbookClick(
                                                TObject *Sender)
{

    Variant my_workbook;
    Variant my_worksheet;
    Variant my_range;

    PropertyGet     Range("Range");
    PropertySet     SetValue("Value");
    PropertySet     SetFormula("Formula");
    PropertyGet     GetValue("Value");
    PropertyGet     GetFormula("Formula");

    my_workbook  = my_excel.OlePropertyGet("ActiveWorkbook");
```

```
    my_worksheet = my_workbook.OlePropertyGet("ActiveSheet");

    Range.ClearArgs();
    SetValue.ClearArgs();
    my_range    = my_worksheet.Exec(Range << "A1");
    my_range.Exec(SetValue << "My Excel Worksheet");

    Range.ClearArgs();
    my_range    = my_worksheet.Exec(Range << "B1:B5");
    my_range.Exec(SetFormula << "=rand()");

    Range.ClearArgs();
    SetFormula.ClearArgs();
    my_range    = my_worksheet.Exec(Range << "A2");
    my_range.Exec(SetFormula << "=sum(b1:b5)");

    ShowMessage(my_range.OlePropertyGet("Value"));
    ShowMessage(my_range.OlePropertyGet ("Formula"));
}
```

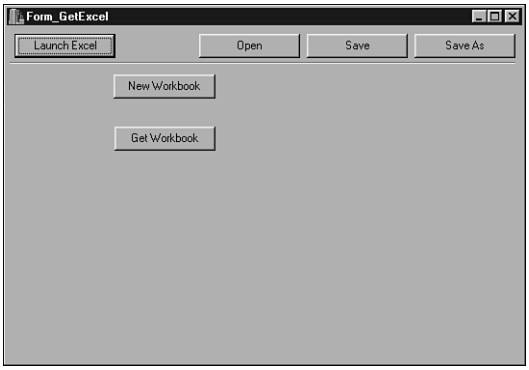


Рис. 21.8. Окно программы `GetExcel`

Основное отличие между свойствами `value` и `formula` — в формате данных, заносимых в ячейку. Свойство `value` требует передачи значения (числового или символьного), а свойство `formula` — передачи текстовой строки в формате представления формулы, так, как это определено в документации программы Excel.

Существуют и другие способы занесения информации в ячейку, например с помощью свойства `text`, которое очень походит на свойство `value`, но отличается от него автоматическим форматированием значений, введенных в текстовом виде. Лично я рекомендую использовать свойство `text` во всех случаях, когда необходимо, чтобы введенные данные рассматривались именно как текст, а не как числа. Если занести в ячейку значение `12345` с помощью свойства `value`, то в ячейке будет запомнено число 12345, независимо от того, желаете ли вы чтобы это был текст (т.е. последовательность символов `"12345"`) или нет.

В структуре объекта `range` имеются также свойства и методы для настройки представления данных из ячейки (или ячеек) на экране. В частности, можно не только изменять параметры шрифта, но и организовать окаймление нескольких ячеек, изменение цвета и т.д.

Свойство `font` объекта `range` возвращает объект типа `font`, который, в свою очередь, имеет свойства, задающие цвет (`color`), начертание (`bold`, `italic`, `shadow` и т.д.) и размер символов.

При форматировании ячеек можно воспользоваться коллекцией `borders`, свойствами `height` и `width`, коллекцией `interior`. Эти объекты и свойства определяют, соответственно, тип окантовки ячеек, их размеры, цвет и образец заливки фона.

Извлечение данных из ячеек рабочего листа

Предлагаю вам внимательнее присмотреться к двум последним операторам в листинге 21.19. Эти операторы не вставляют никакой информации в рабочий лист Excel, а используют метод `OlePropertyGet` для считывания данных из ячейки.

```
ShowMessage(my_range.OlePropertyGet("Value"));
```

Таким способом можно извлекать не только значение из ячейки, но и формулу. Поэтому по такому принципу можно организовать в приложении просмотр содержимого ячеек.

Использование серверных компонентов C++Builder 5

До сих пор мы рассматривали в основном использование в комплексных приложениях механизма автоматизации OLE — OleAutomation. Но помимо этого среда C++Builder 5 позволяет использовать и библиотеку шаблонов ActiveX — ATL (ActiveX Template Library). Доступ к ней открывается через компонент `OleServer` и производные от него компоненты. Они размещены на вкладке Servers палитры компонентов и здесь вы сможете найти несколько компонентов. предназначенных для работы с Microsoft Word и Microsoft Excel (рис. 21.9).

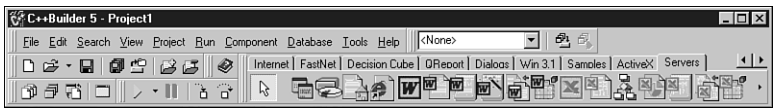


Рис. 21.9. Вкладка Servers палитры компонентов

Компоненты этой группы позволяют быстро интегрировать приложения Microsoft Office в программу, создаваемую в среде C++Builder. Для этого требуется перетащить нужные компоненты на поле экранной формы программы и включить в программу необходимый для управления ими код. Для того чтобы продемонстрировать методику работы с этими компонентами, мы создадим на их базе новую версию программы составления словаря.

Но прежде чем приступить к созданию новой программы, кратко познакомьтесь с некоторыми из компонентов группы Server. Убедитесь, что флажок Code Completion (Завершение кода) в диалоговом окне Environment Options установлен, и создайте новое приложение в среде C++Builder. Перетащите на поле главной экранной формы приложения компоненты `TWordApplication` и `TWordDocument`. Включите в экранную форму новую кнопку, а затем дважды щелкните на ней и отредактируйте связанный с ней программный код.

Как только откроется окно редактора программного кода, введите текст **`WordApplication1->`** и подождите, пока функция завершения кода *Code Completion* “придумает”, как его завершить.

Задумавшись на пару секунд, C++Builder выведет на экран список подходящих методов и свойств (рис. 21.10).

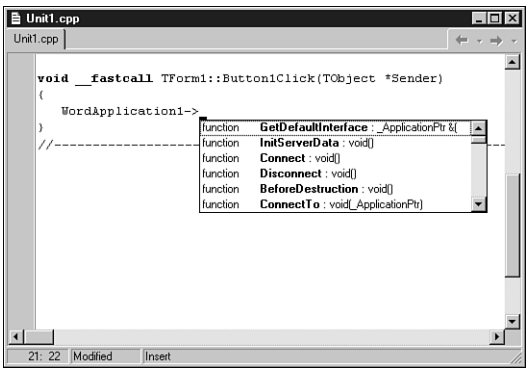


Рис. 21.10. Функции и свойства компонента `TWordApplication`

Просмотрев выведенный список, вы убедитесь, что в нем остались многие функции и свойства объекта приложения, использовавшиеся при работе с OleAutomation. Но учтите, что некоторые свойства и методы несут иную функциональную нагрузку, для других — изменены имена, а в третьих — изменился список аргументов.

Компоненты TWordApplication и TWordDocument

Компонент `TWordApplication` представляет собой ядро приложения Word, т.е. по существу тот же самый объект приложения. который мы использовали в предыдущих разделах этой главы. Правда, `TWordApplication` — более сложный класс, но зато он обеспечивает более тесную интеграцию создаваемой программы с приложением Word. Программа получит более широкие возможности доступа к внутренним функциям Word, что позволит сократить количество промежуточных операций и обеспечит более строгую проверку корректности программного кода на стадии компиляции. При использовании OleAutomation все некорректности "выплывают" только на стадии выполнения программы.

Но учтите, при использовании компонента объем создаваемой программы увеличивается, по сравнению с использованием OleAutomation, а процесс компиляции занимает больше времени, поскольку к программе предъявляются более жесткие требования. Точно так же, как компонент `TWordApplication` является эквивалентом объекта `application` приложения Word, так и компонент `TWordDocument` представляет собой ATL-версию объекта документа Word (объекта `document`). Этот компонент имеет те же достоинства и недостатки, что и компонент `TWordApplication`.

Приняв во внимание все эти предостережения, приступим к созданию новой версии программы составления словаря на основе выбранного документа Word.

Новая версия программы составления словаря

Я предлагаю не "изобретать велосипед", а просто отредактировать фрагменты ранее созданной программы. Начнем с фрагмента, который отвечает за запуск Word, а затем займемся извлечением текста из документа, созданием нового документа и вставкой в него текста. Для взаимодействия с приложением Word будем использовать компоненты `TWordApplication` и `TWordDocument`.

Полный комплект файлов новой версии программы вы найдете на прилагаемом к книге компакт-диске (папка `VocabB`, файл проекта `vocab_proj.bpr`).

Запуск Word

Первым делом в программе нужно организовать подключение к приложению Microsoft Word и взять "бразды правлениям" этим приложением в свои руки. В предыдущих программах мы проверяли, запущено ли приложение Word на компьютере, и либо подключались к уже работающей программе, либо самостоятельно создавали новый объект приложения. В данном случае для выполнения тех же операций будет использоваться метод `connect()`, что позволит избежать промежуточных операций (листинг 21.20).

Листинг 21.20. Запуск Word

```
/*
||
|| Функция PrepareWord
||
*/
void __fastcall TVocabForm::PrepareWord(void)
{
    try
    {
        WordApplication1->Connect();
    }
    catch(...)
    {
        ShowMessage("Unable to load Word");
        // "Не могу загрузить Word"
    WordApplication1->Visible = True;
}
```

Алгоритм выполнения операции остался по существу тем же, что и в предыдущей версии, но его программная реализация значительно упростилась и стала более "прозрачной". Если не включать в программу проверки ошибок времени выполнения (а этого я вам настоятельно не советую делать), то все сведется к двум операторам.

Метод `Connect()`, который использован для подключения к приложению Word, является одним из наиболее часто используемых в компонентах `OleServer` и производных от него. Этот метод обращается к операционной системе и выясняет, работает ли в данный момент определенный сервер Ole Automation (в нашем случае — Word), и либо захватывает контроль над ним, либо самостоятельно создает новый экземпляр сервера и запускает его на выполнение.

Установка свойства `Visible` позволяет убедиться воочию, что приложение Word работает. Как видно из приведенного фрагмента кода, доступ к свойствам объекта можно получить, используя привычный синтаксис C++, а не через промежуточные функции типа `OlePropertySet()`.

Извлечение текста из документа Word

Для того чтобы работать с определенным документом Word, нам понадобится не только компонент `TWordApplication`, но и компонент `TWordDocument`, который будет представлять в нашей программе открытый документ.

Для открытия документа используется метод `Open()` коллекции `documents`, и в этом новая программа практически ничем не отличается от прежней (листинг 21.21).

Листинг 21.21. Извлечение текста из документа Word

```
void __fastcall TVocabForm::Button_VocabClick(TObject *Sender)
{
    OleVariant FileName;
    wchar_t *doc_contents;
    RangePtr my_range;

    PrepareWord();

    if (OpenDialog1->Execute())
    {
        FileName = OpenDialog1->FileName;

        //-- Получено имя файла - открыть документ.
        WordDocument1->ConnectTo(
               WordApplication1->Documents->Open(FileName));
        WordDocument1->Select();

        //-- Извлечь слова из документа.
        my_range = WordDocument1->Range(EmptyParam,EmptyParam);
        doc_contents = my_range->get_Text();

        //-- Передать слова в буфер элемента управления
        //-- типа ListBox и организовать сортировку.
        ListBox_Words->Sorted  = false;
        ListBox_Words->Visible = false;
        ListBox_Words->Clear();

        TransferWordsToList(doc_contents,ListBox_Words);

        //-- Удалить повторяющиеся слова.
        RemoveDuplicateWordsFromList(ListBox_Words);

        //-- Вывести список снова на экран.
        ListBox_Words->Visible = true;

        //-- Создать новый документ.
        WordDocument1->ConnectTo(
                    WordApplication1->Documents->Add());

        //-- Вставить в документ список слов.
        ReportWordsFromList(ListBox_Words);
    }
}
```

На первый взгляд, этот программный код мало чем отличается от первой версии. Главное отличие в том, что значительно сократилось количество промежуточных операций. В этом фрагменте используются всего три переменные для хранения промежуточных результатов и объектов вместо множества переменных типа `Variant` в предыдущей версии.

Процесс извлечения текста из документа Word стал в программе более прозрачным. Бо́льшая часть операторов в листинге 21.21 имеет отношение не столько к извлечению текста из документа, сколько к формированию и сортировке списка слов. А в предыдущем варианте все было как раз наоборот — большинство операторов "занималось" извлечением текста и управлением приложением Word.

Весь процесс загрузки документа Word "уложился" в один оператор:

```
WordDocument1->ConnectTo(
              WordApplication1->Documents->Open(FileName));
```

Все в этом операторе понятно, потому что логично. Извлекается коллекция `documents` непосредственно из объекта компонента `TWordApplication`, и вызывается метод `Open()` этого объекта коллекции, который и загружает документ. Результат используется в качестве аргумента `ConnectTo()` метода компонента `TWordDocument`, и мы сразу получаем доступ к содержимому документа.

Конечно, можно разбить этот оператор на несколько функций. Функция ***Code Completion*** даст *в процессе* разработки программного кода полную информацию о приемлемых для данного объекта свойствах и методах, а также типах аргументов методов. При работе с методами Ole Automation все аргументы имели тип `Variant`, что, с одной стороны, позволяет не задумываться о типе в процессе разработки кода, но, с другой стороны, не страхует от ошибок, которые затем "выползают" наружу при выполнении программы. Новый вариант программы позволяет воспользоваться механизмом проверки типов, который имеется в компиляторе, и, таким образом, выявить и устранить большинство ошибок еще на стадии разработки программного кода.

Открыв документ, мы выбираем его содержимое с помощью метода `Select()`, а затем копируем его в текстовый буфер. Как и прежде, при этом используется объект `range`.

```
my_range = WordDocument1->Range(EmptyParam, EmptyParam);
```

Константа `EmptyParam` эквивалентна передаче функции в качестве аргумента значения `Null`. В данном случае это позволяет извлечь содержимое всего документа целиком в единственный объект `range`.

Далее используется метод `get_Text()` объекта `range`, который и извлекает собственно текст в текстовый буфер. Как видите, это одна из ситуаций, в которой имя метода претерпело некоторые изменения, по сравнению с прежним вариантом программы.

```
doc_contents = my_range->get_Text();
```

Метод `get_Text()` требует особого внимания. Дело в том, что он возвращает текст не в обычном формате, а в формате расширенных символов. Это не страшно, но забывать о таком изменении формата не следует. Если вдруг окажется, что содержимое документа на экране приобрело вдруг непонятный вид, то скорее всего ошибку следует искать именно в использовании результатов метода `get_Text()`.

В нашей программе текст не трансформируется. Он просто передается в метод `TransferWordsToList()`, который принимает аргумент типа `AnsiString`, а далее C++Builder сортирует элементы списка.

Вставка текста в документ

Итак, вас ждет еще одна приятная неожиданность: вставка текста в документ в новом варианте программы выполняется так же просто, как и извлечение.

Функция вставки текста вызывается в программе, представленной в листинге 21.21. Перед вызовом этой функции создается новый документ Word, а потом уже вызывается `ReportWordsFromList()`, и в новый документ вставляются слова из списка.

Создание нового документа выполняется одним оператором:

```
WordDocument1->ConnectTo(WordApplication1->Documents->Add());
```

Как и при открытии существующего документа, в данном случае используется компонент `TWordApplication`, извлекается ссылка на его коллекцию `documents`, а затем вызывается метод `Add()` этой коллекции. Для документа, созданного этим оператором, используется шаблон по умолчанию. Метод `Add()` возвращает дескриптор документа, но мы не будем возиться с ним, а сразу же передаем дескриптор в качестве аргумента методу `ConnectTo()` все того же объекта `TWordDocument`. После этого можно вызывать функцию `ReportWordsFromList()` (листинг 21.22).

Листинг 21.22. Вставка текста в документ Word

```
void __fastcall TVocabForm::ReportWordsFromList(TListBox *lb)
{
    long words;

    WordDocument1->Range(EmptyParam,EmptyParam)->InsertAfter(
          StringToOleStr("Words from '" +
                  OpenDialog1->FileName + "'\n\n"));

    for (words=0;words<lb->Items->Count;words++)
    {
        WordDocument1->Range(
           EmptyParam, EmptyParam)->InsertAfter(
              StringToOleStr(lb->Items->Strings[words]+"\n"));
    }
}
```

Вставка выполняется с помощью метода `InsertBefore()` или `InsertAfter()` объекта `range` (надеюсь, в этом нет ничего для вас неожиданного).

```
WordDocument1->Range(EmptyParam,EmptyParam)->InsertAfter(
        StringToOleStr(lb->Items->Strings[words]+"\n"));
```

Как видите, в этой операции нет ничего сложного. Не забудьте о символе конца строки после каждого вставляемого слова. Когда я в первый раз запустил программу на выполнение, забыв вставить в нее символ конца строки, получился документ, прочесть который было сложновато, поскольку все слова слились в одну строку.

В этой версии не использовался объект типа `table`. Одна из причин состоит в том, что программный код формирования таблицы с помощью компонента `TWordDocument` оказывается несколько сложнее, чем код, выполняющий аналогичные функции при использовании Ole Automation.

Дело в том, что компонент `TWordDocument` не располагает методами, которые позволили бы с минимальными усилиями преобразовать текст в таблицу. Это один из недостатков, характерных для многих компонентов ATL.

Заключительные замечания относительно ATL и серверов OLE

Как и при использовании других инструментальных средств для разработки приложений, использование серверов OLE и компонентов ATL имеет преимущества и недостатки. Поскольку такие компонентами формируют более "тесные" связи с интегрируемыми приложениями, у разработчика появляется возможность получить доступ к множеству функций, недоступных при другой методике интеграции. Кроме того, программный код значительно упрощается и становится "прозрачным", а сама программа работает быстрее.

Однако технология применения серверов OLE и компонентов ATL создает дополнительную нагрузку для компилятора C++Builder и увеличивает объем выполняемой программы. Случается, что более тесные связи приводят к снижению гибкости комбинированного приложения. При установке C++Builder 5 программа инсталляции задает пользователю вопрос, собирается ли он использовать в своей работе серверы OLE для Office 97 или Office 2000. Хотя компонент `TWordDocument`, который мы активно эксплуатировали в приводимых примерах, в равной степени "чувствует себя своим" при работе с любой версией Microsoft Office, этого нельзя сказать о множестве других компонентов.

После того как вы наберетесь определенного опыта использования разных технологий создания интегрированных приложений, вы будете без особого труда выбирать ту из них, которая более соответствует условиям конкретной задачи.

Возможности интегрированных приложений на базе Microsoft Office

Мы рассмотрели простейшие способы интеграции компонентов Microsoft Office, причем только двух из всего набора, в приложения C++Builder. Поэтому очень многое осталось "за кадром".

Word

Ниже приведен список проектов, которые позволят расширить возможности использования Microsoft Word.

- *База данных управления документами*.

  Эта программа импортирует все слова из документов, удаляет самые распространенные (такие как союзы, предлоги, артикли, модальные глаголы и т.п.), а остальные помещает в базу данных. В результате можно быстро отыскать документ, "процитировав" характерные для него слова.

  Эту идею можно развить таким образом, что программа будет просматривать и сетевые каталоги и автоматически обновлять базу данных. Это вовлечет в сферу деятельности программы большинство пользователей сети.

- *Библиотека отчетов*.

  Создайте библиотеку типовых программ для формирования отчетов с помощью Word. Вместо того чтобы использовать для формирования отчетов компоненты типа `QReport`, можно обратиться к Word и получить отчеты в формате, позволяющем их легко распространять.

При использовании Word не только упрощается распространение отчетов, но и расширяются возможности форматирования и компоновки документа. Можно ведь реализовать и такую возможность, как преобразование в формат HTML и обновление содержимого Web-страниц.

- *Программа управления корреспонденцией*.

  Хотя подключение к электронной почте реализовано практически во всех программах обработки текстов, форматы многих из них несовместимы с источниками данных, которые желательно использовать, а часть таких программ не обладает всеми необходимыми функциональными возможностями. Располагая средствами непосредственного управления приложением Word, можно отслеживать поля в документах и автоматически их заполнять. Благодаря этому пользователь может не только хранить копии документов, рассылаемых каждому адресату, но и отслеживать, когда, кому и зачем был отослан тот или иной документ.

Excel

Возможности работы с Excel, которые были продемонстрированы в примерах данной главы, можно расширить — вставлять в ячейки рабочих листов графические диаграммы и базы данных и использовать их затем для представления или накопления информации.

Я полагаю что на базе интеграции с Excel можно создать следующие приложения.

- *Программа анализа журнала посещений Web-страницы*.

  Программа импортирует данные из журнала посещений Web-страницы и использует их для обновления содержимого столбцов электронной таблицы. В результате можно организовать анализ потока посетителей выбранной Web-страницы. Очень хорошо использовать для этого рабочую книгу, на одном листе которой фиксировать первичные данные из журнала, на другом — итоговые данные с заданной периодичностью, на третьей организовать диаграммы, на четвертой — отчеты и т.д.

- *Вопросник с накоплением данных*.

  Если ваша работа связана с накоплением больших объемов информации, подумайте над тем, не использовать ли на каждый вопрос отдельный рабочий лист, в который можно заносить ответы, а затем по специальной программе импортировать такие листы и формировать резюме.

- *Список "Что нужно сделать"*.

  Если почувствуете, что для организации своего времени вам требуется список планируемых работ, подумайте, не сделать ли его на базе Excel. Каждая строка в рабочем листе может представлять отдельную задачу, а ее ячейки — текущее состояние задачи (закончена, вот-вот будет завершена, выполняется и т.п.), планируемую дату завершения и т.д. Можно разработать программу, которая будет просматривать записи и напоминать вам о тех или иных срочных делах.

Другие приложения Office

В состав Microsoft Office входят не только Word и Excel — там есть еще и другие весьма интересные и полезные приложения. Некоторые из них также могут быть интегрированы в приложение, созданное в среде C++Builder, причем для интеграции используются те же технологии, которые мы рассматривали применительно к Word и Excel.

Outlook

Помимо того, что Outlook является программой работы с электронной почтой, это приложение может использоваться и в качестве бизнес-организатора, секретаря и менеджера персональной информации. Объект приложения этой программы носит наименование `outlook.application`.

В отличие от Word и Excel, Outlook обычно не извлекает коллекции или объекты. а использует методы для формирования ссылок на объекты. Например, в листинге 21.23 представлена программа, которая с помощью Outlook формирует новое почтовое сообщение.

Листинг 21.23. Формирование нового почтового сообщения с помощью Outlook

```
Variant my_outlook = Variant::CreateObject("outlook.application");
Variant my_message = my_outlook.OleFunction("CreateItem",(Variant) 0);
    //-- 0 - новый элемент
my_message.OleProcedure ("Display");
```

Access

Строго говоря, Access является наименее приемлемым кандидатом для интеграции в приложение C++Builder. С помощью C++Builder можно значительно лучше организовать работу с базой данных, чем с помощью Access, но последняя является идеальным инструментом для быстрого создания небольших локальных баз данных.

Тем не менее и в отношении Access можно использовать технологию создания интегрированных приложений. С помощью C++Builder несложно организовать передачу информации из баз данных Access в более сложные базы данных. Объект приложения для Access можно получить с помощью `access.application`. Я могу предложить следующие интегрированные приложения на базе Access.

- *Система сбора информации о базе данных*.

  Поскольку Access является весьма удобным инструментом для формирования прототипа базы данных, ее можно использовать для получения полного описания компонентов базы данных — структуры таблиц и запросов. Пользуясь Access вручную, эту информацию можно извлечь, перебирая компоненты поодиночке. В то же время приложение C++Builder может самостоятельно открывать все объекты такой базы данных, извлекать нужную информацию и формировать отчет.

- *Приложение-марионетка*.

  Если ваши заказчики давно пользуются базами данных Access и не рвутся отказываться от привычного инструмента, предложите им программу, которая будет использовать Access в качестве средства работы с конечным пользователем, а вся обработка будет выполняться в фоновом режиме программой, созданной в среде C++Builder.

Project

Microsoft Project в действительности не является компонентом Office, но также обладает возможностью интеграции в другое приложение. Создав с помощью `msproject.application` экземпляр объекта приложения, можно получить доступ к коллекции проектов, ресурсов, задач и т.п. Все эти объекты содержат информацию, которая идеально подходит для связывания с другой системой через C++Builder.

Можно предложить следующие идеи для воплощения с помощью Microsoft Project и C++Builder.

- *Менеджер коллективного решения задачи*.

  Каждый член группы, занимающейся определенным проектом, может иметь программу, которая будет напоминать ему, что от него в настоящее время требуется. Эта программа может вести журнал регистрации поручений с фиксацией даты выдачи задания, контрольного срока получения результата и отметки о текущем состоянии задания.

  Пользуясь своей программой, руководитель проекта всегда может быть в курсе текущего состояния дел и знать, кто чем занимается, в каком состоянии отдельные виды работ, где возможен срыв сроков.

- *Динамическое отображение хода работы над проектом*.

  В среде C++Builder можно разработать программу, которая будет обращаться к Microsoft Project, извлекать информацию о плане выполнения проекта и помещать их в базу данных. Затем можно организовать четкое отображение динамик процесса выполнения проекта при разных условиях.

- *Система отслеживания затрат*.

  Microsoft Project позволяет отслеживать стоимость выполнения отдельных этапов работы над проектом. В программе на C++ можно извлекать эту информацию и использовать ее для выписки и рассылки счетов и других финансовых документов.

Связь с другими приложениями

Приложений, способных работать с OLE Automation, становится все больше. Для их интеграции в программы, разработанные в среде C++Builder, можно использовать ту же методику, которая была продемонстрирована в предыдущих разделах этой главы.

Некоторые из таких приложений перечислены ниже:

- ***FrontPage*** (`frontpage.application`);
- ***Visio*** (`visio.application`);
- ***Internet Explorer*** (`internetexplorer.application`);
- ***Adobe Photoshop*** (`photoshop.application`);
- ***Fusion*** (`fusion.application`).

Резюме

В этой главе были рассмотрены методы создания интегрированных приложений в среде C++Builder, которые позволяют использовать компоненты Microsoft Office. В частности, были продемонстрированы способы управления программами Word и Excel из программы на C++, а также возможности, которые открывает такая интеграция перед разработчиками.

Хотя подробно были рассмотрены только методы интеграции Word и Excel, эта технология применима и к другим приложениям, поддерживающим работу с OLE Automation.

Глава

22

Использование технологии ActiveX

Боб Сворт
Дэймон Чандлер

В этой главе будут описаны компоненты Active Server Pages (ASP) и Active Server Objects (ASO), средства поддержки работы с этими компонентами, имеющиеся в C++Builder, и методика их использования. Вы узнаете, что собой представляют объекты ActiveForms, как их можно создавать в среде C++Builder и внедрять в приложение, позволяя другим пользователям работать с активными формами. Объекты ActiveForms, разработка которых описана в этой главе, будут выводиться в окне Internet Explorer. Но мы расскажем и о том, как их зарегистрировать и использовать в другой среде. При описании методики внедрения ActiveForms мы также рассмотрим и вопросы работы с CAB-файлами и пакетами.

Далее вы узнаете, каким образом преобразовать объект ActiveForm в приложение "тонкого" клиента, удалив из него слой данных и заменив его компонентом ClientDataSet. В результате объект ActiveForm будет преобразован в клиента MIDAS. В этом разделе главы мы вновь вернемся к MIDAS-серверу `CustomerOrders`, разработка которого была описана в главе 19.

При описании методики работы с Active Server Pages и Active Server Objects предполагается. что вы знакомы с технологией программирования Web-приложений, описанной в главе 13.

В заключительном разделе этой главы будет описана работа с программными оболочками и использование при этом структур и функций Windows API и интерфейсов COM.

Основные характеристики объектов Active Server Objects

Компонент ASP (Active Server Pages) представляет собой одно из программных средств поддержки Web-серверных приложений, как, впрочем, и интерфейс CGI (Common Gateway Interface), и расширения API ISAPI/NSAPI (Internet Server API/Netscape Server API). Это означает, что программист может, включив компоненты Active Server Pages и объекты Active Server Objects в состав Web-сервера, предоставить клиенту возможность подключиться к такому Web-серверу и загрузить страницы или объекты. В первой части этой главы основное внимание будет уделено созданию объектов Active Server Objects и их использованию в составе Active Server Pages. Дополнительную информацию о компонентах Active Server Pages можно получить в разделах оперативной справки MSDN, посвященных отдельным аспектам работы с ASP.

В комплект C++Builder 5 включен новый мастер, который окажет программисту помощь при создании объектов Active Server Objects. Такие объекты ASO можно использовать в составе Active Server Pages (ASP) для динамического формирования HTML-кода при каждой очередной загрузке объекта страницы сервером. В этом разделе мы рассмотрим, что собой представляют объекты Active Server Page, как они соотносятся с CGI, ISAPI и моделью COM и как использовать перечисленные средства в контексте Active Server Pages. Далее мы перейдем к различным аспектам процесса формирования объектов Active Server Objects (ASO). ASO представляют собой объекты сервера, причем отличия в версиях серверов Internet Information (Web) Servers IIS 3, IIS 4 и IIS 5 могут определять методику работы с этими объектами.

В качестве примера будет создан простой объект Active Server Object и шаблон сценария, а затем показано, как, добавляя определенные методы, можно настроить этот объект и сценарий в соответствии с конкретными требованиями приложения. Затем будет показано, как установить и зарегистрировать объект на Web-сервере. Завершится раздел описанием внедрения новых версий Active Server Objects, их тестирования и отладки.

Использование мастера *Active Server Object Wizard*

Прежде чем приступать к созданию нового объекта Active Server Object, нужно сформировать модуль библиотеки ActiveX Library (модуль ActiveX Library). В комплекте C++Builder 5 имеются средства, позволяющие автоматизировать процесс создания такого модуля.

1. Запустите C++Builder 5 и закройте предлагаемый по умолчанию проект.
2. Выберите в главном меню C++Builder команду File⇨New. В окне хранилища объектов New Items откройте вкладку ActiveX и выберите в ней ярлык ActiveX Library.
3. Сохраните проект типа ActiveX Library в файле `BCB5ASP.BPR`.

Теперь нужно модифицировать созданный модуль таким образом, чтобы в дальнейшем не пришлось внедрять в него пакеты реального времени (в том числе и динамические библиотеки RTL) с помощью объекта Active Server Object. Хотя в результате объем модуля ActiveX Library увеличится, он сможет работать на любой платформе, включая и такие, где отсутствует комплект C++Builder 5.

1. В главном меню C++Builder 5 выберите команду Project⇨Options.
2. В диалоговом окне Project Options, которое появится после этого на экране, откройте вкладку Linker и сбросьте флажок Use Dynamic RTL (использовать динамическую библиотеку RTL).
3. Откройте вкладку Packages и сбросьте флажок Builder with Runtime Packages (Компилировать с пакетами реального времени).

В модуль ActiveX Library можно добавить объект Active Server Object. Для этого необходимо выбрать ярлык Active Server Object на вкладке ActiveX (рис. 22.1).

Рис. 22.1. Ярлык Active Server Object на вкладке ActiveX диалогового окна New Items

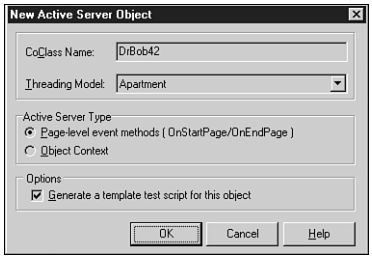


Рис. 22.2. Диалоговое окно мастера New Active Server Object

После этого на экране откроется диалоговое окно мастера New Active Server Object, показанное на рис. 22.2. Для тех читателей, которые видят это окно впервые (в особенности, если у них нет опыта работы с объектами COM или ASP), мы объясним назначение некоторых элементов управления в нем.

В поле CoClass Name вводится внутренне имя формируемого объекта COM. Как правило, выбор имени ничем не ограничивается, и в примерах этой главы мы будем использовать имя `DrBob42`. В списке Threading Model по умолчанию устанавливается вариант `Apartment`,

который идеально подходит для объектов ASO. Эту настройку менять не следует. Описание других вариантов модели потоков вы найдете в главе 16.

Наибольший интерес представляет переключатель Active Server Type. Выбор настройки этого переключателя зависит от того, какая версия Internet Information (Web) Server установлена на вашем компьютере (или на вашем Web-сервере). При работе с IIS 3 и 4 используются методы обработки событий на уровне страницы в сочетании с обработчиками `OnStartPage()` и `OnEndPage()`, в то время как IIS 4 и IIS 5 позволяют использовать метод контекста объектов (Object Context), т.е. управлять экземплярами объектов Active Server Object с помощью Microsoft Transaction Server (MTS). C++Builder 5 инкапсулирует большинство отличий между этими вариантами, а потому можно оставить без изменений предлагаемый по умолчанию вариант Page-level events methods (с объектом Active Server Objects можно будет выполнять практически те же операции и при выборе варианта Object Context). Но не забывайте о том, что выбор варианта должен учитывать тип Web-сервера, установленного на вашем компьютере, иначе он работать не будет.

При установке флажка Generate a template test script for this object C++Builder сформирует довольно простой HTML-сценарий тестирования созданного объекта Active Server Object. Если вы недостаточно хорошо знакомы с языком составления сценариев для ASP, эта опция вам поможет изучить его. В сформированном C++Builder сценарии будет всего пара строк, но они продемонстрируют, как вызывать методы объекта ASO. Короче говоря, при настройке в этом диалоговом окне вам придется ввести самостоятельно только имя сопряженного класса в поле CoClass Name, а все прочие элементы управления можно не трогать, оставив предлагаемые по умолчанию настройки. После ввода **`DrBob42`** в поле CoClass Name щелкните на кнопке OK.

Редактор библиотеки типов

По завершении манипуляций в диалоговом окне New Active Server Object будет создан объект ASO, включающий модуль библиотеки типов. Последняя процедура — настройка библиотеки типов объекта `DrBob42` в окне редактора библиотеки типов (рис. 22.3).

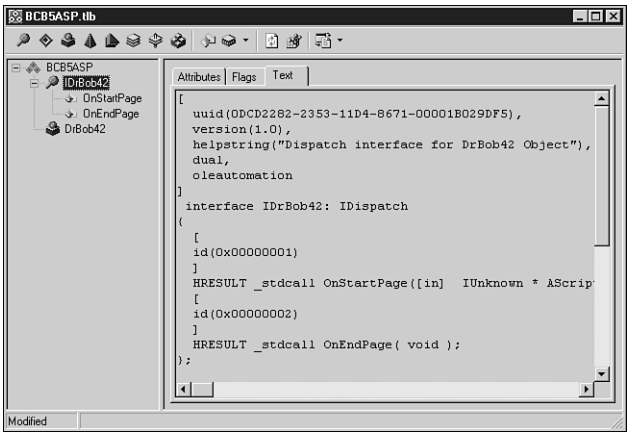


Рис. 22.3. Окно редактора библиотеки типов

Учтите, что при новом сохранении проекта C++Builder предложит сохранить и файл HTML-шаблона ASP (для данного объекта он будет называться `DRBOB42.ASP`). В этом файле содержится следующий HTML-текст:

```
<HTML>
<BODY>
<TITLE>Testing C++Builder ASP </TITLE>
<CENTER>
<H3>You should see the results of your
C++Builder Active Server method below </H3>
</CENTER>
<HR>
<%Set CBuilderASPObj =Server.CreateObject("BCB5ASP.DrBob42")
CBuilderASPObj.{Сюда вставьте имя метода}
%>
<HR>
</BODY>
</HTML>
```

В тэгах ASP используется символ `%`, который отличает их от обычных HTML-тэгов. В единственном ASP-тэге содержится сценарий из двух строк. В первой строке формируется экземпляр объекта `DrBob42` из модуля библиотеки ActiveX `BCB5ASP`, а во второй — вызывается метод, причем имя метода пропущено и вам придется вписать его самостоятельно.

На рис. 22.3 обратите внимание на такой нюанс: в интерфейс `IDrBob42` объекта уже включены методы `OnStartPage()` и `OnEndPage()`. Это сделано в соответствии с выбором варианта Page-Level Event Methods в диалоговом окне New Active Server Object. Как обычно, реализация методов интерфейса библиотеки типов размещается в модулях исходного кода объекта — в данном случае объекта Active Server Object (листинг 22.1).

Листинг 22.1. Класс `TDrBob42Impl`: программный код объекта Active Server Object

```
//DRBOB42IMPL :реализация  TDrBob42Impl
#include <vcl.h>
#pragma hdrstop
#include "DRBOB42IMPL.H"
#include "WebMod.h"
////////////////////////////////////////////////////////////
//TDrBob42Impl
STDMETHODIMP TDrBob42Impl::OnEndPage()
{
   HRESULT hr =E_FAIL;
   try
   {
      hr =TASPObject::OnEndPage();
   }
   catch(Exception &e)
   {
      return Error(e.Message.c_str(),IID_IDrBob42);
   }
   return hr;
}

STDMETHODIMP TDrBob42Impl::OnStartPage(LPUNKNOWN AScriptingContext)
{
   HRESULT hr =E_FAIL;
```

```
   try
   {
      hr =TASPObject::OnStartPage(AScriptingContext);
   }
   catch(Exception &e)
   {
      return Error(e.Message.c_str(),IID_IDrBob42);
   }
   return hr;
}
```

В составе класса можно определить и другие методы, помимо "стандартных", — `OnEndPage()` и `OnStartPage()`. Например, с помощью библиотеки типов в интерфейс `IDrBob42` можно добавить метод `TheAnswer()` (для этого нужно щелкнуть правой кнопкой мыши на узле `IDrBob42` в левой панели окна редактора библиотеки типов и выбрать в контекстном меню команду New⇨Method). Этот метод мы будем использовать для вывода приветствия. После добавления метода в состав интерфейса и обновления нужно разработать программный код реализации метода в сопряженном классе `TDrBob42Impl::TheAnswer()`. Но прежде вам необходимо познакомиться с некоторыми особенностями структуры внутренних объектов ASP и средствами C++Builder 5, которые помогают программировать такого рода объекты. Как и приложения WebBroker, ASP предлагает специальные объекты типов `Request` и `Response`.

ASP-объект Response

Объект `Response` является внутренним для ASP, причем доступ к нему возможен через методы интерфейса объектов ASO. `Response` следует использовать в тех случаях, когда возникает необходимость в динамическом выводе. В структуре объекта `Response` имеется множество свойств и методов, среди которых особое положение занимает метод `Write()`. Этот метод принимает в качестве аргумента переменную типа `Variant` и обеспечивает динамический вывод специфицированного аргумента (в том месте сценария ASP, где задано обращение к этому методу). Для записи приветствия вставьте в метод `TDrBob42Impl::TheAnswer()` программный код, приведенный в листинге 22.2.

Листинг 22.2. Использование метода `Write()`

```
STDMETHODIMP TDrBob42Impl::TheAnswer()
{
   HRESULT hr =E_FAIL;
   try
   {
      Response->Write(Variant("Hello Bob,"));
      Response->Write(Variant(",<P>"));
      Response->Write(Variant(
             "The answer to the question about Live,"));
      Response->Write(Variant(
             "The Universe and Everything is...."));
      Response->Write(Variant("<BR>"));
      hr =Response->Write(Variant("<B>42</B>"));
   }
```

```
   catch(Exception &e)
   {
      return Error(e.Message.c_str(),IID_IDrBob42);
   }
   return hr;
}
```

В файле DRBOB42.ASP в тэг ASP нужно внести единственное изменение (имя метода — TheAnswer()). В новой редакции тэг ASP будет выглядеть следующим образом:

```
<%Set CBuilderASPObj = Server.CreateObject("BCB5ASP.DrBob42")
CBuilderASPObj.TheAnswer
%>
```

Этим практически исчерпывается подготовка первого теста объекта Active Server Object в составе Active Server Page. Теперь нужно зарегистрировать объект ASO BCB5ASP.OCX и разместить DRBOB42.ASP в том подкаталоге, на который распространяются права сценария ASP. На выполнение всех этих операций уйдут считанные минуты.

ASP-объект Request

Еще один объект ASP, который играет важную роль в функционировании страницы, носит наименование Request. Как и Response, объект Request доступен через методы интерфейса объекта Active Server Object. Объект Request используется при считывании поступивших сообщений. Существует три способа задания входных сообщений: через переменные Form при помощи метода POST, через переменные URL при помощи метода GET, и с помощью вспомогательных объектов cooky. Соответственно, для извлечения сообщений используются методы get_Form(), get_QueryString() и get_Cookies(). Все эти методы входят в состав интерфейса IRequestDictionary.

Например, для считывания значения из переменной Name объекта Form нужно сначала с помощью метода get_Form(&Form) получить ссылку на Form (типа IRequestDictionary), а затем вызвать метод get_Item(), передав ему в качестве первого аргумента наименование переменной, а ее значение — вторым аргументом (листинг 22.3). В этом листинге приведен новый вариант метода TheAnswer(), в котором извлекается значение переменной Name из экранной формы ввода данных, которую можно использовать при запуске сценария ASP.

Листинг 22.3. Модифицированный вариант метода TheAnswer()

```
STDMETHODIMP TDrBob42Impl::TheAnswer()
{
   HRESULT hr =E_FAIL;
   try
   {
      Response->Write(Variant("Hello"));
      Variant Name;
      IRequestDictionary*Form;
      Request->get_Form(&Form);
      Form->get_Item(Variant("Name"),Name);
      Response->Write(Name);
      Response->Write(Variant(",<P>"));
      Response->Write(Variant(
```

```
            "The answer to the question about Live,"));
      Response->Write(Variant(
            "The Universe and Everything is...."));
      Response->Write(Variant("<BR>"));
      hr =Response->Write(Variant("<B>42</B>"));
   }
   catch(Exception &e)
   {
      return Error(e.Message.c_str(),IID_IDrBob42);
   }
   return hr;
}
```

Ту же методику можно использовать и по отношению к объектам `QueryString` и `Cookies`, хотя последний используется довольно редко, поскольку в ASP имеются собственные средства сохранения информации о состоянии.

Запрос объектов ASO

Объекты Active Server Objects можно регистрировать и как встроенный COM-сервер, и как внешний COM-сервер. Первый вариант используется чаще, поскольку в этом случае проще обеспечить защиту сервера. Мы в дальнейшем будем использовать объекты именно в статусе встроенного COM-сервера. На практике часто оказывается, что Web-сервер "отказывается" работать с таким объектом, когда ему придается статус внешнего COM-сервера.

Для регистрации `BCB5ASP.ocx` как встроенного сервера нужно выбрать в главном меню C++Builder команду Run⇨Register ActiveX. Для удаления регистрационной записи того же сервера (необходимость в этом возникает при профилактической очистке системного реестра компьютера) нужно в меню выбрать команду Run⇨Unregister ActiveX Server.

Работа с объектом ASP

Хотя объект Active Server Page можно загрузить и автономно, чаще всего загрузка выполняется в ответ на запрос из HTML-формы. Например, можно использовать такую простенькую HTML-страницу:

```
<HTML>
<HEAD>
<TITLE>Dr.Bob's ASP Examine</TITLE>
</HEAD>
<BODY BGCOLOR=FFFFCC>
<FONT FACE="Verdana" SIZE=2>
Who disturbs my slumber?
<P>
<FORM ACTION="http://192.168.92.201/doc/drbob42.asp" METHOD=POST>
It is me,<INPUT TYPE=TEXT NAME="Name">
<P>
<INPUT TYPE=SUBMIT>
</FORM>
</BODY>
</HTML>
```

После загрузки ее как страницы `http://192.168.92.201/bcb5/drbob42.htm` в Internet Explorer, на экране появится картинка, представленная на рис. 22.4.

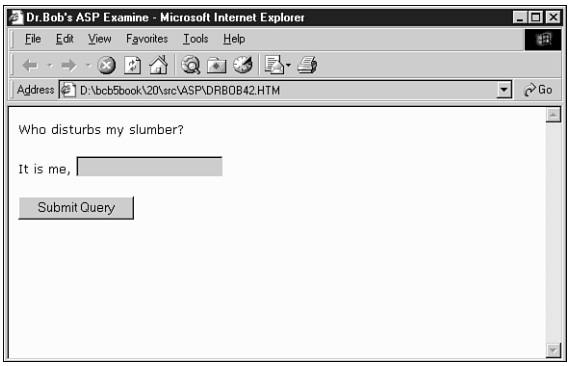


Рис. 22.4. Форма `DRBOB42.HTM` в окне Internet Explorer

После того как пользователь введет свое имя и щелкнет на кнопке Submit Query, будет загружен объект Active Server Page. Он, в свою очередь, в соответствии с ASP-сценарием создаст объект ASO `DrBob42` и вызовет метод `TheAnswer()`. В результате динамически сформируется страница, представленная на рис. 22.5.

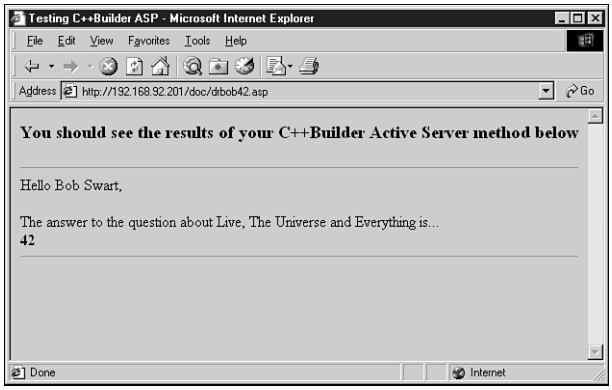


Рис. 22.5. Страница в окне Internet Explorer, динамически сформированная сценарием `DrBob42.asp`

Конечно, это весьма простенький пример использования ASP, в котором применяются только объекты `Request` и `Response`, но этого вполне достаточно для демонстрации основной идеи.

Объекты ASP `Session`, `Server` и `Application`

Объект ASP позволяет получить доступ не только к объектам `Request` и `Response`, но также и к объектам `Session`, `Server` и `Application`. В этом одно из преимуществ ASP перед CGI и ISAPI: объекты Active Server Object могут получить доступ к информации о текущем сеансе и приложении, причем для этого требуются минимальные усилия со стороны программиста.

Напомню, что в рассмотренном выше примере класс TDrBob42Impl сформирован как производный от класса TASPObject. В структуре этого класса, помимо свойств Request и Response, имеются также свойства TASPObject::Session, Server и Application. Через эти свойства можно получить доступ к ASP-объектам Session, Server и Application, соответственно. Эти же свойства можно использовать и в методах TDrBob42Impl (например, для сохранения имен "посетителей" Web-страницы). В HTML-сценарии ASP эта процедура имеет следующий вид (обратите внимание на вторую строку):

```
<%Set CBuilderASPObj=Server.CreateObject("BCB5ASP.DrBob42")
Session.Value("UserName")="Bob Swart"
CBuilderASPObj.TheAnswer
%>
```

Для извлечения этого сохраняемого значения (сохраняемого среди прочих объектов Active Server Pages, которые посещаются тем же пользователем в том же сеансе) можно использовать объект Session в составе объекта Active Server Object, точно так же, как мы использовали объекты Form, QueryString или Cookies. Остается единственный вопрос: каким образом объект Session обрабатывает подобную информацию о состоянии? Для этого используются *cookies* (читается "кукиз"; термин не имеет "официального" перевода на русский язык — *прим. ред.*), т.е. фрагментов информации, полученных клиентом как ответ на запрос и сохраняемых им. Поэтому при настройке объекта ASO не отключайте внутренний механизм работы с *cookies*, иначе в процессе функционирования объекта могут возникнуть проблемы.

Объекты ASP и поддержка WebBroker

Помимо поддержки внутренних функций объектов ASP, в C++Builder 5 имеются и средства поддержки объектов ASO. Для таких объектов можно использовать те же компоненты формирования HTML-кода, которые применялись для WebBroker и описаны в главе 13. Единственным исключением является компонент TQueryTableProducer. Дело в том, что он рассчитан на работу с объектом Request WebBroker, а не с аналогичным объектом ASP. Все прочие компоненты PageProducers можно использовать точно так же, как и при работе с WebBroker. Это будет продемонстрировано в примере, описанном чуть ниже. Добавьте в проект модуль WEBMOD.CPP (его разработка была описана в главе 13) и выберите в главном меню C++Builder команду File⇨Include Unit Hdr. С помощью этой команды добавьте модуль DRBOB42IMPL из предыдущего примера. Теперь в проекте можно будет использовать Web Module из главы 13. Измените программный код метода TheAnswer():

```
STDMETHODIMP TDrBob42Impl::TheAnswer()
{
HRESULT hr =E_FAIL;
try
   {
   Response->Write(Variant("Hello"));
   Variant Name;
   IRequestDictionary* Form;
   Request->get_Form(&Form);
   Form->get_Item(Variant("Name"), Name);
   Response->Write(Name);
   Response->Write(Variant(",<P>"));
   TWebModule1* wm =new TWebModule1(NULL);
   hr = Response->Write(
```

```
        Variant(wm->DataSetTableProducer1->Content()));
    FreeAndNil(wm);
}
catch(Exception &e)
{
    return Error(e.Message.c_str(), IID_IDrBob42);
}
    return hr;
}
```

Обращаю ваше внимание на то, что в этом варианте метода `TDrBob42Impl::TheAnswer()` динамически создается и уничтожается объект `TWebModule1`, который мы "заимствовали" из программы, рассмотренной в главе 13. Дело в том, что в Web Module поступающие запросы обрабатываются средствами самого модуля, а объект Active Server Object не имеет собственных средств работы с очередью поступающих запросов. Другими словами, если в приложение типа ActiveX Library добавлен глобальный модуль Web Module, то любой экземпляр объекта Active Server Object (для каждого запроса) сможет обратиться к нему, по при этом существует потенциальная угроза конфликта множества потоков. Чтобы устранить эту угрозу, Web Module должен иметь статус локального компонента каждого экземпляра Active Server Object. Поскольку он используется только в пределах метода `TDrBob42Impl::TheAnswer()`, вполне логично в процессе выполнения метода динамически создать экземпляр Web Module, а затем его уничтожить.

Если в разрабатываемом объекте ASO нужно организовать множество соединений, то можно создать и собственный пул объектов WebBroker.

Установка объектов ASO

При повторной компиляции объекта Active Server Object компоновщик выдаст сообщение об ошибке `The file is still in use` (Файл уже используется). Можно попытаться остановить Microsoft Internet Information Server, но это вряд ли поможет. Не поможет в этой ситуации и выключение службы ***World Wide Web Publishing Service***. Придется выключить всю службу администрирования ***IIS Admin Service***, и только после этого модуль `ASP.DLL` и все объекты Active Server Objects будут удалены из памяти и появится возможность повторно скомпилировать любой из них. Учтите, что выключение службы ***IIS Admin Service*** приведет к тому, что все зависимые от нее службы (WWW, FTP и т.д.) также будут выключены, а вот этого на работающем Web-сервере делать не хотелось бы. Таким образом, в реальной ситуации процедура установки объекта Active Server Objects может приобрести характер ночного кошмара.

Упростить процесс выключения и перезапуска административных служб NT можно с помощью несложного пакетного файла `RESTART.BAT`, используемого при повторной компиляции и переустановке объекта ASO:

```
net stop "World Wide Web Publishing Service"
net stop "IIS Admin Service"
net start "World Wide Web Publishing Service"
```

По завершении компиляции новой версии объекта Active Server Object нужно повторно загрузить файл `DrBob42.htm` и запустить объект Active Server Page. На экране появится картинка, представленная на рис. 22.6.

Возможность использования компонентов WebBroker в объекте Active Server Object зафиксирована в документации фирмы Borland, поскольку этим значительно расширяется поддержка ASP в среде C++Builder.

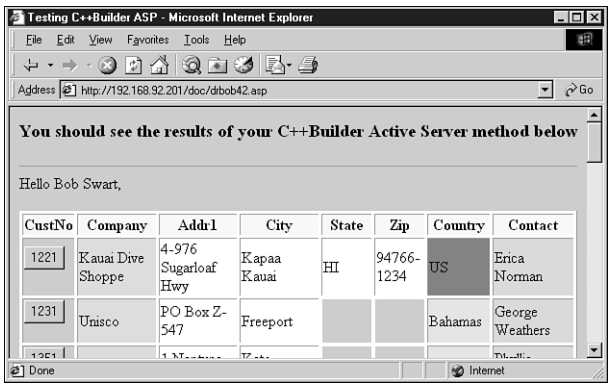


Рис. 22.6. Выход `DrBob42.asp` компонента WebBroker в окне Internet Explorer

Отладка объектов ASO

Как следует из предыдущих разделов, объекты ASO во многом схожи с DLL-модулями ISAPI. После загрузки такого модуля выгрузить его можно, только остановив работу Web-сервера. Это происходит по той причине, что объекты ASO загружаются модулем `ASP.DLL`, который сам по себе является DLL-модулем ISAPI. Однако преимущество ASP состоит в том, что сценарии самих объектов Active Server Page можно обновлять сколько угодно, не перезагружая при этом объектов Active Server Object. До тех пор пока внутренние функции объекта ASO не изменяются, можно обойтись только редактированием сценариев. Но отладка и тестирование объектов ASO — это совсем другая задача.

В главе 36 руководства разработчика *Developer's Guide*, выпущенного фирмой Borland, которое поступает пользователям в комплекте с C++Builder 5, содержится описание процесса отладки объектов ASO. Но, к сожалению, эта инструкция не имеет ничего общего с реальностью. Лично я, сколько ни пытался отлаживать такие объекты, пользуясь этой инструкцией (а это происходило на компьютерах совершенно разных конфигураций, причем речь идет об объектах, разработанных не только в C++Builder, но и в Delphi), так и не смог добиться положительных результатов.

Оказалось, что можно выполнять только самые простые отладочные процедуры — выводить окна сообщений или использовать окно отладки для просмотра сообщений, отсылаемых объектом. Но для того чтобы получить такое сообщение, нужно объявить себя владельцем отлаживаемого объекта, которому разрешено взаимодействовать с рабочим столом (desktop). В частности, при использовании службы ***IIS Admin Service*** установите флажок Interact with Desktop в диалоговом окне Services панели управления системой (Control Panel). Конечно, такой способ отладки не очень удобен, но это все же лучше, чем не иметь никакого.

Объекты ActiveForms

Вы уже имели возможность познакомится с объектами Active Server Pages и Active Server Objects, которые играют большую роль в серверных Web-приложениях, и со средствами программирования таких объектов в среде C++Builder 5. Что касается клиентских Web-приложений, то C++Builder позволяет "упаковать" экранную форму такого приложения в элемент управления ActiveX и "опубликовать" ее в сети Web или включить в другое приложе-

ние, созданное как в среде C++Builder, так и Delphi. Такие элементы управления ActiveX получили наименование объектов типа ActiveForm. Сам по себе объект ActiveForm — это "черный ящик" со строгой внутренней структурой и тщательно продуманным интерфейсом общения с внешним миром. Такие объекты имеет смысл использовать не только в Web-приложениях, но и в качестве элементов технологии *разработки на базе компонентов* (CBD — Component-Based Development).

Создание объекта ActiveForm

Проще всего начинать создание объекта ActiveForm "с чистого листа". Но на практике часто приходится использовать ранее созданное приложение, которое каким-то образом нужно преобразовать в объект ActiveForm. В этом случае на помощь могут прийти фреймы и шаблоны, которые применимы как к обычному приложению, так и к приложению типа ActiveForm.

Для создания ActiveForm-приложения "с чистого листа" выберите в главном меню C++Builder команду File⇨New и в появившемся на экране диалоговом окне New Items откройте вкладку ActiveX. Выберите в ней ярлык мастера Active Form Wizard. Этот мастер позволяет задать имя объекта (по умолчанию предлагается `ActiveFormX`), наименование модуля реализации и библиотеки ActiveX. Важнее всего именно последняя настройка, поскольку именно она определяет наименования элемента управления ActiveX, который будет в конце концов создан. В данном случае (рис. 22.7) выбрано наименование `DRBOB42.OCX`. Обращаю ваше внимание на то, что в поле Threading Model при создании объекта ActiveForm нужно всегда оставлять предлагаемый по умолчанию вариант `Apartment`. Это необходимо для использования в дальнейшем созданного объекта ActiveForm в Internet Explorer 4 или более поздних версий.

В диалоговом окне мастера ***Active Form Wizard*** имеется три флажка. Первый — Make Control Licensed (Формировать лицензируемый элемент управления) — устанавливает режим создания специального файла лицензирования (с расширением `.LIC`). Такая настройка используется при создании объектов ActiveForm, предназначенных для продажи сторонним разработчикам. При наличии такого файла в составе проекта тем, кто будет использовать разработанный объект, придется сначала купить право на его применение.

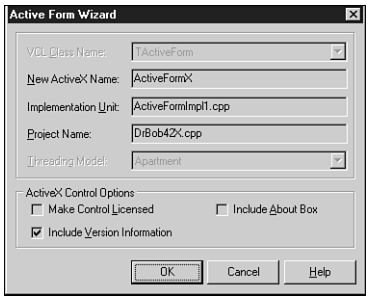


Рис. 22.7. Окно мастера **Active Form Wizard**

Второй флажок — Include About Box (Включать окно About) — устанавливается, если необходимо включать в ActiveForm-приложение возможность просмотра информационного окна About.

Третий флажок — Include Version Information (Включать информацию о версии) — играет наиболее существенную роль. При его установке в состав приложения включается информация о версии. Если такая информация в составе объекта отсутствует, Web-броузер не может определить, следует ли загружать обновленную версию имеющегося в нем объекта. Я рекомендую никогда сбрасывать этот флажок. В диалоговом окне Web Deployment Options имеется аналогичная опция, но когда речь идет об объектах ActiveForms и информации о версии, лучше продублировать эту настройку, чем забыть о ней.

После щелчка на кнопке OK в окне Active Form Wizard появится сообщение о том, что в текущий проект нельзя добавить элемент управления ActiveX, поскольку он не является объектом типа *ActiveX library*. (Это сообщение, естественно, не появится, если на первом этапе

создания приложение выбран тип ActiveX Library.) В окне сообщения щелкните на кнопке OK и приступайте к созданию модуля DLL, который может содержать элемент управления ActiveX. Избежать появления этого сообщения можно и таким способом: закройте все проекты, прежде чем начинать создание нового ActiveForm-приложения, или запустите C++Builder из командной строки с ключом `-np` — эта настройка отменяет создание проекта по умолчанию при запуске C++Builder.

Обратите внимание на то, что модуль ActiveX Library получает расширение `.OCX` — это расширение задается на вкладке Application диалогового окна Project Options и в правиле `PROJECT` в файле проекта.

В диалоговом окне Project Options откройте вкладку Linker и сбросьте в ней флажок Use Dynamic RTL. После этого перейдите на вкладку Packages и сбросьте флажок Build with Runtime Packages. Если не сделать этого, то вместе с объектом ActiveForm придется устанавливать и модули динамической библиотеки времени выполнения и пакета времени выполнения. Наше первое ActiveForm-приложение — совсем простое и не включает дополнительных модулей.

На этой стадии работы можно приступать к включению в состав приложения компонентов и функций. Хочу обратить ваше внимание на то, что создаваемое приложение имеет определенные ограничения — оно не поддерживает работу с главным меню, а окно приложения не имеет строки заголовка. Функции главного меню можно передать панели инструментов или контекстному меню, а строку заголовка можно сымитировать при помощи панели в экранной форме, свойству `Align` которой следует задать значение **`alTop`**, свойству `Color` — значение **`clActiveCaption`**, свойству `Color` шрифта — значение **`clCaptionText`**, а свойству `Style` шрифта — значение **`fsBold`**.

Экранная форма нашего первого приложения ActiveForm представлена на рис. 22.8, и в ней все организовано как можно проще. Позже мы расширим ее функциональные возможности. Помимо панели (компонента `TPanel`), которая используется для имитации строки заголовка, в экранной форме присутствует только одна кнопка. Текст обработчика события `OnClick` этой кнопки приведен ниже:

```
void __fastcall TActiveFormX::Button1Click(TObject *Sender)
{
   ShowMessage("Hello C++Builder 5 ActiveForm");
}
```

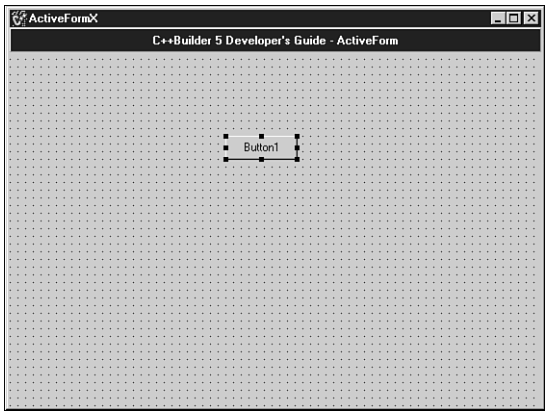


Рис. 22.8. Экранная форма первого ActiveForm-приложения на этапе разработки

Сохраните CPP-модуль реализации приложения и файл проекта на диске (последнему присвойте имя **RBOB42X.DPR**, что приведет в результате к созданию откомпилированного модуля DRBOB42X.OCX). На этом конструирование первого ActiveForm-приложения завершается.

Установка ActiveForm-приложения для использования в Internet Explorer

Теперь перейдем к установке созданного ActiveForm-приложения и его отображению в Internet Explorer версии 3.01 (или более поздней). Эта версия обычно включается в состав операционной системы большинства персональных компьютеров, поэтому с доступом к ней не должно возникать никаких проблем. Если же на вашем компьютере не установлен Internet Explorer, эту программу можно найти на компакт-диске C++Builder 5 или загрузить с Web-сервера Microsoft по адресу http://www.microsoft.com. Независимо от того, устанавливается ли ActiveForm-приложение для работы в глобальной сети Internet, локальной сети Intranet или только на отдельном компьютере, все равно начинать следует с выбора команды Web Deployment Options в меню Project. На экран будет выведено диалоговое окно Web Deployment Options (рис. 22.9). Но прежде чем вызывать это окно на экран, нужно скомпилировать проект.

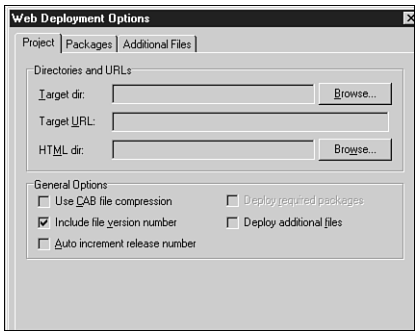


Рис. 22.9. Диалоговое окно Web Deployment Options

Настройка параметров ActiveForm-приложения

В верхней части вкладки Project диалогового окна Web Deployment Options находится зона Directories and URLs с тремя полями настройки:

- Target dir;
- Target URL;
- HTML dir.

В поле Target dir задается каталог, в который будет установлен выполняемый файл ActiveForm-приложения и сопутствующие двоичные файлы. Эти файлы можно распространять по сети Web и передавать любому пользователю, который подключился через эту сеть к вашему компьютеру. При установке ActiveForm-приложения в Internet или локальной сети intranet эта настройка помогает получить доступ к Web-серверу. В такой ситуации, как правило, приложение и сопутствующие файлы устанавливаются в каталог wwwroot или в один из его подкаталогов. Например, можно установить файлы ActiveForm-приложения в каталоги C:\WEBSHARE\WWWROOT, C:\INETPUB\WWWROOT или C:\INETPUB\WWWROOT\MYACTIVEFORMS.

Если же планируется установить ActiveForm-приложение только на отдельный компьютер, можно указать в этом поле любой каталог, например D:\BCB5BOOK\ACTIVEFORMS (именно этот каталог будет использован в примерах, представленных в этой главе).

Адрес URL, указанный в поле Target URL, будет использован в HTML- или INF-файлах (о них чуть ниже), которые запускают ActiveForm-приложение. Строка в этом поле должна указывать на каталог, в котором размещается ActiveForm-приложение, готовое к установке. Как правило этот

каталог задается в формате адреса URL (что-то вроде `http://localhost/MyActiveForms/`), но можно использовать и относительную форму задания URL с помощью `./`. Это означает, что ActiveForm-приложение и его HTML- или INF-файл находятся в том же каталоге, что и Web-сервер. Этот прием срабатывает и в том случае, когда файл ActiveForm-приложения и HTML/INF-файл находятся в одном каталоге на локальном компьютере.

При установке разработанного ActiveForm-приложения компоненты C++Builder формируют для этого проекта простой HTML- или INF-файл. HTML-файл формируется в том случае, если ActiveForm-приложение не обращается к дополнительным файлам (пакетам, динамическим библиотекам RTL и т.д.), а в противном случае создается INF-файл. В данном случае будет создан HTML-файл.

Созданный средствами C++Builder HTML-файл можно загрузить в броузер и использовать его для запуска ActiveForm-приложения. В этот HTML-файл без участия разработчика включаются только те HTML-тэги, которые необходимы для вывода формы приложения. Поэтому в большинстве случаев нужно скопировать созданную C++Builder заготовку в тот HTML-файл, с помощью которого будет реально контролироваться функционирование приложения. В созданной заготовке основной интерес представляет тэг `OBJECT`. Если проект предполагает установку нескольких файлов, то HTML-файл будет ссылаться на второй файл с расширением `.INF` — это тот самый INF-файл, о котором говорилось выше. В INF-файле содержится информация об адресе URL, по которому размещены файл ActiveForm-приложения и прочие дополнительные файлы, необходимые для его функционирования, — файлы пакета, динамической библиотеки RTL и т.п. Короче говоря, адрес URL, указанный в поле Target URL, должен указывать либо непосредственно специфицировать каталог с файлом ActiveForm или каталог с INF-файлом, который, в свою очередь, указывает каталог размещения ActiveForm-приложения и сопутствующих файлов.

Если ActiveForm-приложение устанавливается на Web-сервер по адресу `http://localhost`, причем файл ActiveForm находится в заданном по умолчанию каталоге `wwwroot`, то в поле Target URL нужно задать **`http://localhost/`**. Этот адрес по умолчанию указывает на каталог `wwwroot` на компьютере сервера. Если при тестировании приложения у вас нет доступа к Web-серверу, то задайте имя каталога, указанного в поле Target dir, например **`D:\BCB5BOOK\ACTIVEFORMS\`**. Конечно при этом можно получить доступ к объекту только на том же компьютере или на другом, который имеет отображение диска с файлами ActiveForm-приложения.

В поле HTML dir задается каталог, в котором должен быть размещен HTML- или INF-файл, созданный C++Builder. Как правило, это тот же каталог, что и заданный в поле Target dir, что является еще одним доводом в пользу ввода значения **`./`** в поле Target URL.

На заметку

При работе с диалоговым окном Web Deployment Options вы наверняка обратили внимание на флажок Default в нижней части вкладок этого окна. При установке этого флажка текущие настройки в остальных полях, в том числе и пути, будут скопированы в системный реестр и автоматически использованы при следующем сеансе работы с аналогичным проектом. Это несколько облегчит работу разработчика, который размещает файлы создаваемых ActiveForm-приложений в одном каталоге и использует при этом одни и те же настройки.

После заполнения полей зоны Directories and URLs на вкладке Project диалогового окна Web Deployment Options проверьте, установлен ли флажок Auto Increment Release Number (Автоматическое наращивание номера версии). При установке этого флажка *после* каждого очередного сеанса установки данного ActiveForm-приложения номер версии (третья цифра в последовательности 1.0.0.0) будет автоматически увеличиваться. По завершении настройки диалоговое окно Web Deployment Options должно выглядеть, как на рис. 22.10.

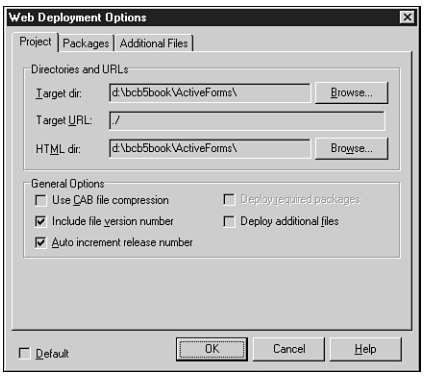


Рис. 22.10. Диалоговое окно Web Deployment Options *после настройки*

На заметку

Обращаю ваше внимание на то, что номер версии изменяется именно *после* очередной установки. После первой установки будет сформирован внутренний номер 1.0.1.0 (загляните на вкладку Version Info в диалоговом окне Project Options). Но пользователи установленной версии увидят номер 1.0.0.0. Поэтому, если вы получите от пользователя сообщение, что в его версии обнаружилась ошибка, то учтите, что это не самая последняя из имеющихся у вас версий приложения, а *предыдущая*. Но если на вкладке Version Info в диалоговом окне Project Options будет установлен флажок Auto-increment Build Number, то изменение номера версии произойдет не после установки приложения, а после его компиляции. Поэтому установленная версия будет иметь тот же номер, что и имеющаяся у вас откомпилированная (номер откомпилированной версии — четвертая цифра).

По завершении настройки закройте диалоговое окно Web Deployment Options и в меню Project выберите команду Web Deploy. Файл скомпилированного ActiveForm-приложения и сформированный C++Builder HTML-файл (или INF-файл и все дополнительные файлы установки) будут автоматически скопированы в каталоги, заданные в окне Web Deployment Options.

На заметку

Если приложение устанавливается с дополнительными файлами библиотек и т.п., то настройка будет выполняться не так просто, как это описано здесь. Придется выполнить кое-какие настройки и на других вкладках диалогового окна Web Deployment Options. Подробно о таком варианте настройки рассказано в разделе *Работа с пакетами и CAB-файлами* далее в этой же главе.

Подключение к ActiveForm-приложению

Теперь можно подключиться к каталогу, в котором хранится ActiveForm-приложение (этого же или другого компьютера), загрузить приложение на компьютер клиента и увидеть его в окне Web-броузера Internet Explorer. Чтобы разобраться в том, что при этом происходит, проанализируем содержимое HTML-файла, сформированного C++Builder:

```
<HTML>
<H1>C++Builder 5 ActiveX Test Page </H1><P>
You should see your C++Builder 5 forms or controls
embedded in the form below.
<HR><center><P>
<OBJECT
classid="clsid:21B973D6-1201-11D4-8669-00001B029DF5 "
```

```
codebase="./DrBob42X.ocx#version=1,0,0,0 "
width=554
height=395
align=center
hspace=0
vspace=0
>
</OBJECT>
</HTML>
```

Идентификатор класса CLSID, представленный в этом файле, представляет собой глобальный уникальный идентификатор (GUID — Globally Unique IDentifier), связанный с объектом. Строка codebase указывает на каталог, в котором размещается файл приложения на компьютере. Частично этот параметр задается в поле Target URL диалогового окна Web Deployment Options. Если приложение выполняется на локальном компьютере, который не является Web-сервером, этот адрес имеет формат спецификации каталога в DOS, а не формат адреса URL. Например, он может иметь форму:

```
d:\bcb5book\ActiveForms\DrBob42X.ocx#version=1,0,0,0
```

Если файл приложения находится в том же каталоге, что и HTML- или INF-файл (т.е. в поле Target URL задано ./), то в строке codebase будет следующий адрес:

```
./DrBob42X.ocx#version=1,0,0,0
```

Теперь можно запустить броузер на компьютере и указать ему адрес URL, с которого нужно загрузить HTML-файл:

```
http://localhost/ActiveForms/DrBob42X.htm
```

Конечно, на вашем компьютере файл может находиться и по другому адресу. Например, можно задать адрес загрузки, используя в URL префикс file:///:

```
file:///d:/bcb5book/ActiveForms/DrBob42X.htm
```

Если экранная форма приложения не выведена в окно Internet Explorer, проверьте следующие настройки.

- Убедитесь, что используется версия Microsoft Internet Explorer 3.0x или более поздняя. Если это не так, значит, проблемная ситуация возникла из-за того, что установленная версия броузера не поддерживает работу с элементами управления ActiveX.
- Если все же используется достаточно "свежая" версия Internet Explorer, выберите в его меню команду View⇨Options и откройте в диалоговом окне Options вкладку Security. Установите в группе Security Level (Уровень безопасности) переключатель Medium (Средний). Если используется Internet Explorer 4 или более поздней версии, то в этой группе нужно установить переключатель Low. Следует установить максимальный уровень безопасности, который допускает работу с элементами управления ActiveX в данной версии Internet Explorer.
- Убедитесь, что флажки Runtime Packages и Dynamic RTL были сброшены при создании ActiveForm-приложения (это — первое, на что я обратил ваше внимание при создании проекта ActiveForm-приложения).

При обращении к элементам управления в броузере скорее всего появятся вопросы о коде подписи (code signing). В окне такого сообщения щелкните на кнопке Yes — после этого можно будет загрузить ActiveForm-приложение. Если у вас возникнет желание воспользоваться средствами подписи, загляните на сервер Microsoft по адресу http://www.microsoft.com/ и познакомьтесь с документами, относящимися к теме *Authenticode* и *Security*.

Создание ActiveForm-приложения, работающего с данными

В качестве следующего примера разработанное ранее приложение будет модифицировано таким образом, чтобы в нем можно было использовать пакеты. Это, в общем-то, настройка, предлагаемая C++Builder по умолчанию, но мы изменили ее при создании первого приложения. Теперь придется вернуться на вкладку Packages диалогового окна Project Options и установить флажок Builder with Runtime Packages.

В дальнейшем мы преобразуем ActiveForm-приложение в интеллектуального ("толстого" — fat) клиента с помощью BDE, таблиц и прочих компонентов. Учтите, что обычно для этого используются модули данных, но в рассматриваемом примере мы будем добавлять элементы управления, работающие с данными, непосредственно на поле экранной формы приложения.

Откройте в среде C++Builder 5 созданный ранее проект ActiveForm-приложения и перетащите на поле формы два компонента `TTable`, назовите их `TableCustomer` и `TableOrders`, установите в их свойствах `Alias` значение **`BCDEMOS`**, а в качестве значений свойств `TableNames` укажите таблицы **`CUSTOMER.DB`** и **`ORDERS.DB`**. Перетащите на поле формы два компонента `TDataSource` и подключите их к `TableCustomer` и `TableOrders`, соответственно. Воспользовавшись редактором полей (он вызывается с помощью щелчка на каждой таблице правой кнопкой мыши), выберите те поля, которые планируется выводить в экранную форму. Для представления в экранной форме полей главной таблицы (таблицы клиентов) используйте компонент `TDBEdits`, а для полей подчиненной таблицы заказов — компонент `TDBGrid`.

Для того чтобы указать, что таблица `TableOrders` будет подчиненной по отношению к таблице `TableCustomer`, воспользуйтесь окном Object Inspector и установите в свойстве `MasterSource` объекта `TableOrders` ссылку на `DataSource` объекта `TableCustomer`. После этого щелкните на свойстве `MasterFields` в окне Object Inspector и задайте поля, связывающие обе таблицы. В качестве индекса выберите `CustNo` и свяжите поля `CustNo` главной и подчиненной таблиц. На рис. 22.11 показано, как выглядит экранная форма приложения на этапе разработки.

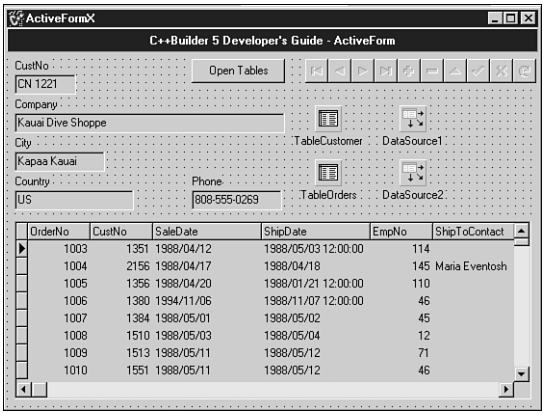


Рис. 22.11. Экранная форма работающего с данными ActiveForm-приложения на этапе разработки

Обращаю ваше внимание на то, что свойство `Caption` объекта кнопки `Button1` изменено на **`Open Tables`**. Это сделано для того, чтобы во время разработки отключить свойство `Active` таблиц в экранной форме.

Обе таблицы открываются после щелчка на этой кнопке. Программный код обработчика события OnClick приведен ниже.

```
void __fastcall TActiveFormX::Button1Click(TObject *Sender)
{
   TableCustomer->Active = true;
   TableOrders->Active = true;
}
```

Теперь проверим, как ActiveForm-приложение будет компилироваться с использованием пакета реального времени. Откройте диалоговое окно Project Options (команда меню Project⇨Options), а затем — его вкладку Packages и установите флажок Add Run-Time Packages. После этого скомпилируйте ActiveForm-приложение и убедитесь, что новая настройка воспринята.

Следующая фаза разработки вновь потребует манипуляций в диалоговом окне Web Deployment Options. На этот раз нужно задать опции для установки пакета реального времени. Сначала установите флажок Deploy Required Packages на вкладке Project. Этот флажок был сброшен перед компиляцией предыдущей версии приложения. Установите также и флажок Use CAB File Compression, который задает режим сжатия и файла ActiveForm-приложения, и всех сопутствующих файлов (CAB — аббревиатура от Cabinet). Сжатие файлов несколько ускорит процесс загрузки, но потребуется время на распаковку. Другим словами, сжатие имеет смысл "заказывать" в том случае, когда ожидается, что файлы приложения будут иметь внушительные размеры. На рис. 22.12 показано, как должно выглядеть окно Web Deployment Option по завершении настройки.

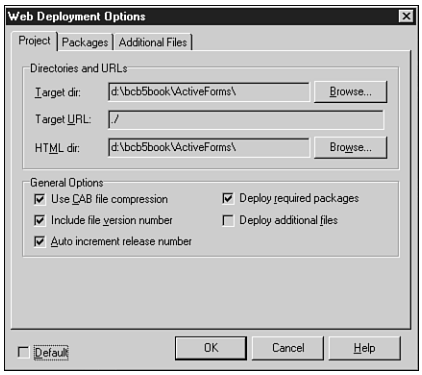


Рис. 22.12. Вкладка Project диалогового окна Web Deployment Options после настройки новой версии ActiveForm-приложения

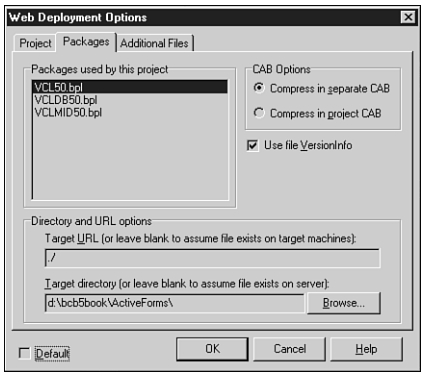


Рис. 22.13. Вкладка Packages диалогового окна Web Deployment Options

Теперь откройте вкладку Packages (рис. 22.13), на которой выполняется настройка параметров установки пакетов реального времени.

На этой вкладке имеется список необходимых для работы разрабатываемого ActiveForm-приложения пакетов, найденных программой C++Builder. Для каждого пакета можно указать целевой адрес URL и целевой каталог установки (эти поля можно и не заполнять, если файлы уже существуют на том компьютере или сервере, где планируется установить приложение).

На этом настройка параметров установки завершается; диалоговое окно Web Deployment Options можно закрыть и приступить к собственно установке. ActiveForm-приложения. Процедура установки запускается после выбора в меню C++Builder команды Project⇨Web

Deploy. Сам процесс займет некоторое время — несколько минут на не очень быстродействующем компьютере, — поскольку перед установкой потребуется сформировать CAB-файл, а уже потом копировать его в указанный каталог.

В диалоговом окне Web Deployment Options имеется возможность настроить такой режим создания приложения, когда все необходимые файлы помещаются в один большой CAB-файл или в набор определенных CAB-файлов. Подумайте над тем, не лучше ли распределить элементы управления или пакеты по разным CAB-файлам; Это позволит сымитировать процесс установки довольно сложного приложения. В таких приложениях можно обновлять в процессе тестирования и отладки не все компоненты, а только отдельные, заменяя выбранные CAB-файлы.

Работа с пакетами и CAB-файлами

Использование в составе ActiveForm-приложения и пакетов, и CAB-файлов на первый взгляд может показаться весьма странным решением, но при определенных обстоятельствах оно может казаться вполне оправданным. Об этом методе распространения приложения имеет смысл подумать, если пользователями приложения являются участники довольно компактной группы в пределах локальной сети intranet. После загрузки одним из пользователей первого полного комплекта ActiveForm-приложения, включающего, помимо самого приложения, пакеты и модули DLL, остальные члены группы будут нуждаться только в файле самого приложения, а загружать сопутствующие файлы им не понадобится. Поэтому можно будет до минимума сократить объем загружаемого ActiveForm-приложения. Такой подход имеет смысл особенно в тех случаях, когда в одну организацию или группу организаций поставляется несколько похожих ActiveForm-приложений.

Но если планируется широко распространять создаваемое ActiveForm-приложение, то имеет смысл организовать его в виде единого файла без дополнительных файлов пакетов и динамических библиотек RTL, поскольку объем загружаемой по сети информации сокращается. Например модуль C++Builder RTL (`CP3245MT.DLL`) имеет объем около 900 Кбайт (и требует еще и модуля `BORLNDMM.DLL` объемом в 24 Кбайт), а объем пакета `VCL50.BPL` около 1,8 Мбайт. Это значительная нагрузка на сеть, особенно, если пользователь загружает приложение по каналу с низкой пропускной способностью, например через модем. Конечно совсем необязательно создавать ActiveForm-приложение, которое использовало бы и динамические модули RTL, и пакеты, но одни редко работают без других.

На заметку

Использование пакетов, как правило снижает требования к объему памяти, поскольку часто используемый код помещается в специальный DLL-модуль. Например, даже в простенькой программе средства работы с базами данных требуют от 100 до 200 Кбайт. Если устанавливать шесть программ такого типа у одного пользователя, то в каждой из них эти 200 Кбайт будут “висеть” ненужным грузом. При использовании пакетов все повторяющиеся в приложениях компоненты можно свести в один пакет и хранить только одну его копию на все шесть приложений, что позволит сэкономить около 1 Мбайт дискового пространства.

Обновление ActiveForm-приложения

После повторной загрузки ActiveForm-приложения (файла `D:\BCB5BOOK\ACTIVEFORMS\DRBOB42X.HTM`) в окне Internet Explorer будет выведена прежняя версия. Это одна из проблем с ActiveForm-приложениями, удовлетворительное решение которой до сих пор не найдено. Если такое случилось с вами, закройте Internet Explorer и попробуйте запустить его вновь. Если этот нехитрый прием не поможет, повторите процедуру установки (но помните, что ранее

был установлен флажок Auto Increment Release Number в окне Web Deployment Options и, следовательно, номер версии изменится).

Если и это не поможет, перейдите в каталог OCCACHE (в Windows 95) или Downloaded Program Files (подкаталог основного каталога Windows) и отыщите в нем файл DRBOB42X.OCX. Удалите этот файл (при этом программа Internet Explorer не должна быть активной) и попробуйте повторить эксперимент.

Каталоги OCCACHE и Downloaded Program Files

Только что вам на собственном опыте пришлось убедиться в том, что повторная установка ActiveForm-приложения далеко не всегда проходит гладко. В операционных системах Windows 95 и Windows 98 существует только один надежный способ выгрузки из памяти элемента управления ActiveX — перезагрузить операционную систему. Другими словами, при попытке повторно установить элемент управления ActiveX вы всегда можете столкнуться с ситуацией, когда прежний вариант DLL-модуля этого элемента не удален из памяти. Хотя в документации на операционную систему и утверждается, что модуль DLL выгружается из памяти по завершении работы с ним, это, мягко говоря, не соответствует действительности. Некоторые особо "продвинутые" пользователи в таких случаях используют специальные программы выгрузки DLL, другие применяют к особо "строптивому" DLL-модулю сначала функцию LoadLibrary(), а затем дважды вызывают FreeLibrary().

Как правило, файлы элементов управления ActiveX, загружаемые в компьютер по сети, сохраняются в каталоге OCCACHE, который является подкаталогом корневого каталога WINDOWS или WINNT в операционных системах Windows 95 и Windows NT 4. (В операционных системах Windows 98, Windows 2000 и тех версиях Windows 95, которые обновлены при установке Internet Explorer 4, эту роль выполняют подкаталоги Downloaded Files или Downloaded Program Files. Иногда в этих подкаталогах организуются собственные подкаталоги — что-то вроде CONFLICT.1, CONFLICT.2 и т.д.

Если в процессе разработки программы вам понадобится создать режим "чистого клиента", то, во-первых, перезагрузите операционную систему, а во-вторых, удалите из указанного подкаталога все подозрительные файлы прежней версии отлаживаемой программы. Кроме того, при установке приложений некоторые файлы могут быть помещены и в каталоги WINDOWS\SYSTEM или WINNT\SYSTEM32. Не поленитесь заглянуть в эти каталоги и посмотреть, не остались ли в них файлы от прежнего сеанса установки. Отыщите их, удалите из реестра и из компьютера. Только теперь можно с уверенностью сказать, что ваш компьютер действительно имитирует "чистого" клиента. Конечно, работая с каталогами, нужно быть особенно осторожным, поскольку случайно можно провести слишком уж радикальную хирургическую операцию и удалить и пакеты, которые нужны для работы самой программы C++Builder.

Очень важно проводить тестирование создаваемого приложения именно на "чистом" компьютере. Только так можно воспроизвести реальную ситуацию, которая существует на компьютерах ваших будущих пользователей.

Представьте себе, что случится, если тестируемый код использует пакет, установленный по умолчанию на вашем компьютере, но не входит в комплект поставки и, следовательно, отсутствует на компьютере потенциального пользователя. Именно это происходит с некоторыми поставщиками упрощенных версий приложений ActiveX, которые невнимательно относятся к комплектованию поставки и забывают включить в него какой-нибудь пакет или DLL. В результате пользователь приобретает программу, устанавливает ее на своем компьютере, пытается запустить и ... "фокус не удался". Дальнейший сценарий очень прост: впредь пользователь не обратится к этому поставщику и найдет себе другого, продукция которого не допускает подобных "проколов".

Постарайтесь, чтобы с вами такого не произошло.

Всегда старайтесь проверять работу созданного ActiveForm-приложения на абсолютно "чистом" компьютере. Пусть, если это возможно, на нем будет установлена только операционная система. Не стоит пренебрегать этими рекомендациями.

ActiveForm-приложение в качестве клиента MIDAS

В этом разделе мы поэкспериментируем с MIDAS-сервером, описанным в главе 19 (для работы вам понадобится редакция Enterprise продукта C++Builder), и преобразуем интеллектуальное клиентское ActiveForm-приложение ("толстого" клиента) в совершенно "тонкого" клиента, который более не будет нуждаться в средствах ядра BDE на компьютере, где находится броузер. Выполните следующие операции.

1. Удалите из приложения компоненты `TableCustomer` и `TableOrders`. При этом не забудьте о соответствующей директиве включения в заголовочном файле: `#include <DBTables.hpp>`. В противном случае приложение по-прежнему будет требовать присутствия BDE.
2. Перетащите на поле экранной формы приложения два компонента two `ClientDataSet` и один — `DCOMConnection` из вкладки Midas палитры компонентов. Компонент `DCOMConnection` нужен для организации подключения к серверу MIDAS, описанному в главе 19.
3. В окне Object Inspector откройте список в поле значения свойства `ServerName` нового компонента `DCOMConnection`. В этом списке перечисляются все серверы MIDAS, доступные на данном компьютере. Если в нем отсутствует `MidasServer.CustomerOrders`, то либо вернитесь к главе 19 и, следуя приведенным там инструкциям, создайте это серверное приложение, либо разработайте собственное. Есть и еще один вариант: скопируйте проект сервера с прилагаемого к книге компакт-диска (он находится в папке `MIDAS`, файл проекта `MidasServer.bpr`), скомпилируйте и запустите программу на выполнение. При первом сеансе выполнения программа регистрируется в системном реестре. После регистрации соответствующий элемент появится в списке доступных серверов MIDAS, и его можно будет использовать при настройке компонента `DCOMConnection`.
4. После настройки свойства `ServerName` можно установить в свойстве `Connected` компонента `DCOMConnection` значение **`true`** и проверить правильность подключения в серверу MIDAS. Если соединение существует, на экране появится окно сервера.
5. Закройте приложение сервера MIDAS (установите в свойстве `Connected` значение **`false`**) и продолжайте работу над проектом ActiveForm.
6. Выделите первый из установленных в приложении компонентов `ClientDataSet`. В его свойстве `RemoteServer` установите ссылку на компонент `DCOMConnection`.
7. В окне Object Inspector откройте список доступных значений свойства `ProviderName` этого компонента `ClientDataSet`. Скорее всего в нем будет только один элемент — `DataSetProviderCustomerOrders`, который и нужно выбрать.
8. Теперь компонент `DataSource1` (в котором ранее был установлен указатель на таблицу `TableCustomer`) свяжите с настроенным компонентом `ClientDataSet` и задайте свойству `ClientDataSet.Active` значение **`true`**. Это позволяет уже на стадии разработки просмотреть в экранной форме реальные данные из главной таблицы.

Теперь перейдем к настройке подчиненной таблицы в экранной форме. Подключите второй компонент `ClientDataSet` к `DataSetField` первого компонента `ClientDataSet` (подробно эта процедура расписана в главе 19).

1. Добавьте `DataSetField` в качестве сохраняемого (persistent) поля первого компонента `ClientDataSet` — щелкните на `ClientDataSet` правой кнопкой мыши и выберите в контекстном меню команду Fields. Добавьте с помощью редактора полей все поля, включая и вложенный набор записей типа `DataSetField` из таблицы `TableOrders`.
2. Выделите второй компонент `ClientDataSet` и свяжите его свойство `DataSetField` с объектом `ClientDataSet1.TableOrders` (в данном случае это также единственный доступный вариант).
3. Подключите второй компонент `DataSource` к этому компоненту `ClientDataSet` и задайте его свойству `Active` значение **`true`**. Проверьте правильность вывода реальных данных из таблицы. На рис. 22.14 показано, как выглядит экранная форма приложения в процессе разработки.

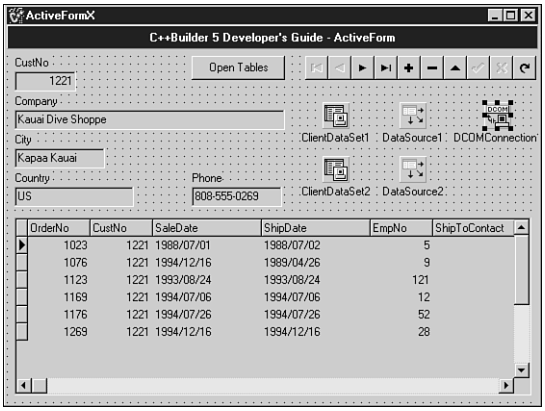


Рис. 22.14. Экранная форма ActiveForm-приложения, работающего с сервером MIDAS, на этапе разработки

Осталась последняя операция — отредактировать код в обработчике события `Button1Click()`. Поскольку теперь в приложении нет таблиц `TableCustomer` и `TableOrders`, вместо них следует активизировать два компонента `ClientDataSet`:

```
void __fastcall TActiveFormX::Button1Click(TObject *Sender)
{
   ClientDataSet1->Active =true;
   ClientDataSet2->Active =true;
}
```

Обратите внимание на то, что вам не пришлось формировать отношение "главный–подчиненный" между двумя компонентами `ClientDataSet` на стороне клиента — это самостоятельно сделает сервер MIDAS. После установки и запуска на выполнение созданного ActiveForm-приложения результат его работы будет выглядеть на экране, как на рис. 22.15. То же самое происходило и в прежней версии приложения, но в данном случае на клиентском компьютере не размещается ядро BDE (правда, на нем должны размещаться некоторые средства поддержки MIDAS, о которых подробно рассказано в главе 19).

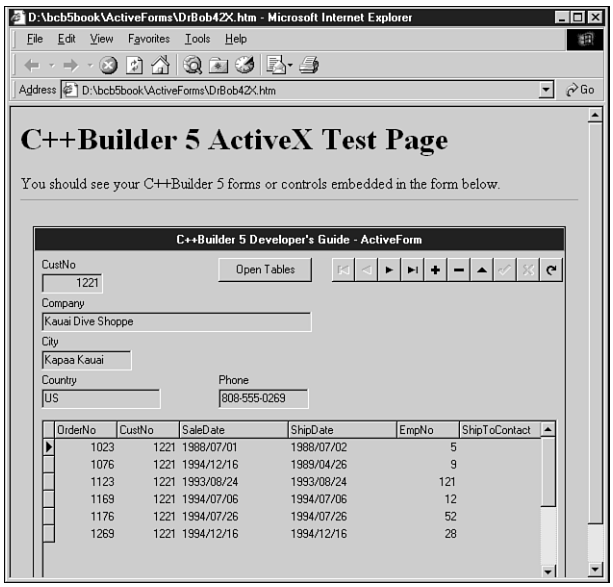


Рис. 22.15. ActiveForm-приложение, работающее с сервером MIDAS, в окне Internet Explorer

Использование ActiveForm-приложения в среде Delphi

Иногда возникает необходимость использовать ActiveForm-приложение в качестве компонента программы, разработанной в среде Delphi или Visual Basic, или в качестве компонента такого приложения как Word. В этом случае нужно только скомпилировать и скомпоновать приложение и проверить, зарегистрировано ли оно в системном реестре. Учтите, что в таком варианте ActiveForm-приложения не следует использовать пакетов времени выполнения, а потому перед началом компиляции сбросьте соответствующий флажок настройки.

Ниже будет продемонстрировано, как импортировать ActiveForm-программу в приложение Delphi 5. Запустите Delphi 5 и в меню Component выберите команду Import ActiveX Control. В диалоговом окне Import ActiveX щелкните на кнопке Add и выберите в списке файл DRBOB42X.OCX, который находится в одном из каталогов локального диска компьютера (рис. 22.16). Вы увидите в этом окне не только имя файла, но и имя класса соответствующего элемента управления ActiveX. При желании можно изменить вкладку палитры компонентов, в которую будет установлен новый компонент (по умолчанию он устанавливается в ActiveX). Щелкните на кнопке Install и ActiveForm-приложение будет установлено как компонент вкладки ActiveX палитры компонентов Delphi 5.

Теперь можно перетащить этот компонент на поле экранной формы нового приложения Delphi (рис. 22.17) и использовать, как “родной” компонент этой среды. Но учтите, что в среде Delphi ничего в разработанном ActiveForm-компоненте менять нельзя, т.е. нельзя получить доступ к отдельным элементам управления в ActiveForm. Если возникнет необходимость в какой-либо “перестройке”, придется вернуться в среду C++Builder и там внести изменения. Новый ActiveForm-компонент взаимодействует с Delphi-приложением только через те свой-

ства и события, которые представлены в окне Object Inspector. Если нужно расширить возможности интерфейса, то придется, вернувшись в C++Builder, добавить при помощи редактора библиотеки типов в интерфейс новые свойства, события или методы. Учтите, что не нужно выполнять процедуру установки ActiveForm-приложения для того, чтобы использовать его в Delphi, Visual Basic или Word, хотя при желании вы можете это сделать.

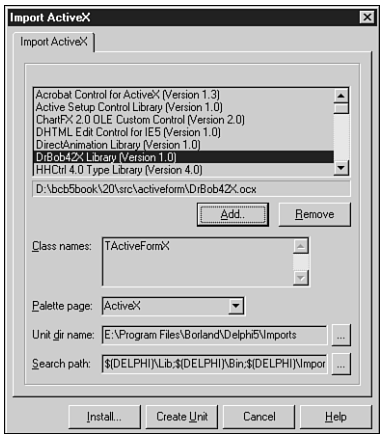


Рис. 22.16. Диалоговое окно Import ActiveX среды разработки Delphi 5

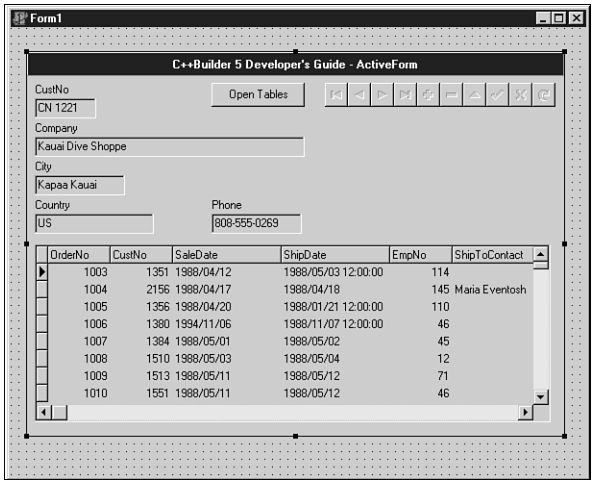


Рис. 22.17. Экранная форма приложения Delphi 5, в которой установлен ActiveForm-компонент BCB5

Создание шаблонов ActiveForm-приложений

Иногда требуется преобразовать разработанную ранее "обычную" программу в ActiveForm-приложение. Среда C++Builder поможет выполнить эту процедуру. Главная операция в такой процедуре — преобразование экранной формы в активный шаблон. Откройте экранную форму исходного приложения в среде C++Builder и с помощью мыши выделите все компоненты в этой экранной форме. После этого откройте меню Component и выберите в нем команду Create Component Template. C++Builder сформирует новый шаблон, в который будут включены все выделенные компоненты экранной формы и связанный с ними программный код.

На заметку

Должен вас предупредить о некоторых ограничениях, которые действуют в отношении шаблонов компонентов. Весь программный код обработки событий сохраняется в шаблонах компонентов, но определенные пользователем методы не сохраняются. Пусть, например, в исходном приложении обработчик события `OnClick` какой-либо кнопки вызывает метод `Go()` класса экранной формы. Код самого обработчика события `OnClick` будет сохранен в шаблоне, а код метода `Go()` — нет. В результате, когда этот шаблон компонента будет включен в ActiveForm-приложение, он не будет выполнять весь набор операций, запрограммированных в исходном приложении (причем, скорее всего, его даже не удастся скомпилировать). Поэтому при подготовке исходного приложения к преобразованию сосредоточьте весь необходимый программный код только в обработчиках событий компонентов экранной формы.

Кроме того, в обычном приложении существует множество событий самого класса экранной формы — `OnShow`, `OnHide`, `OnClose`, `OnCloseQuery`, `OnResize`, `OnCanResize`, `OnDock` и `OnMouseWheel`, которые отсутствуют в классе ActiveForm. Программный код, который включен в обработчики этих событий, должен быть распределен по другим обработчикам, которые сохраняются в классе ActiveForm.

Для использования созданных шаблонов компонентов нужно выполнить следующие операции.

1. В главном меню C++Builder выберите команду File⇨New, а затем на вкладке ActiveX выделите ярлык Active Form. При этом программа C++Builder предложит создать новую библиотеку, с чем следует согласиться.
2. После этого откройте вкладку Templates палитры компонентов и выделите в ней шаблон, который был создан ранее из обычного приложения. Перенесите его на поле экранной формы создаваемого ActiveForm-приложения.
3. Сохраните файлы проекта и скомпилируйте новое приложение. Это все, что требуется сделать для того, чтобы на основе ранее созданного обычного приложения сформировать ActiveForm-приложение.

И последнее замечание: версия C++Builder 5 поддерживает концепцию фреймов. *Фрейм* — это коллекция компонентов, во многом сходная с шаблоном компонентов, но обладающая большей гибкостью в настройке при включении ее в состав ActiveForm-приложения. Фреймы можно преобразовывать в шаблоны. Если вы начинаете новый проект и перед вами стоит задача создать приложение в двух вариантах — обычного приложения и ActiveForm-приложения, — то использование фреймов позволит включить один и тот же блок пользовательского интерфейса в оба варианта.

Программирование оболочек

Интерфейс прикладного программирования оболочек (shell API) включает набор структур, функций и COM-интерфейсов, которые могут быть использованы в программных оболочках, работающих в операционной среде Windows. Как и другие компоненты службы интерфейса пользователя ***User Interface Services***, API-оболочки разрабатываются с тем, чтобы предоставить пользователю возможность управлять стандартными средствами операционной системы Windows и расширять их функциональные возможности.

До изобретения программных оболочек Windows все операции с файловой системой требовали манипуляций на уровне "физических" объектов файлов. Если требовалось отыскать в файловой системе определенный файл, к нему можно было добраться, только задав абсолютный путь. Пусть, например, вам необходимо отыскать местонахождение файла `autoexec.bat`. Для этого потребуется просмотреть список файлов каждого каталога и проанализировать совпадение элементов списка с заданным именем файла.

Хотя этот подход можно реализовать и в 32-разрядной операционной системе, идентификация определенного файла по его абсолютному пути никак нас удовлетворить не может. Задумайтесь, например, над тем, как отыскать фал приложения ***Control Panel***. Какой каталог нужно просматривать? Конечно же не `C:\CONTROLPANEL`. Фактически ***Control Panel*** не является "физическим" объектом файловой системы, содержимым которого можно манипулировать с помощью обычных средств. Такие объекты принято называть *виртуальными* (*virtual object*), и обслуживаются они программной оболочкой. Среди других часто используемых виртуальных объектов следует упомянуть ***My Computer***, ***Printers*** и ***Dial-Up Networking***. Некоторые виртуальные объекты могут храниться и на удаленных компьютерах — к таким, в ча-

стности, относятся объекты сетевых принтеров. Именно для поддержки операций с такими виртуальными объектами в программной оболочке организуется каталог всех объектов файловой системы (как физических, так и виртуальных), причем каждому из них присваивается идентификатор SHITEMID, а все идентификаторы помещаются в список ITEMIDLIST.

Использование идентификаторов объектов

На концептуальном уровне идентификатор объекта файловой системы можно рассматривать как аналог имени файла или имени папки, а список идентификаторов — как аналог абсолютного пути к файлу или папке. Но при формировании идентификаторов программная оболочка использует не символьные строки, а специальный структурный тип SHITEMID, определенный в файле SHLOBJ.H:

```
typedef struct _SHITEMID
{
   USHORT cb;
   BYTE abID [1];
}SHITEMID;

typedef UNALIGNED SHITEMID *LPSHITEMID;
typedef const UNALIGNED SHITEMID *LPCSHITEMID;
```

В состав структурного типа SHITEMID входит два члена: короткая беззнаковая целая переменная cb и массив байт abID, имеющий переменную длину. Переменная cb содержит длину массива abID. Собственно идентификатор объекта содержится в массиве abID, причем он является уникальным в пределах "родительской" папки. Можно сопоставить каждому идентификатору (переменной типа SHITEMID) имя файла, которое также является уникальным в пределах определенной папки, но в других папках именованных пространств могут содержаться файлы с таким же именем.

Можно провести аналогию и между списком идентификаторов (это переменная структурного типа ITEMIDLIST), и абсолютным путем в файловой системе, который совместно с именем файла уникально характеризует каждый отдельный объект. Определение структурного типа ITEMIDLIST приведено ниже (это определение включено в файл SHLOBJ.H).

```
typedef struct _ITEMIDLIST
{
   SHITEMID mkid;
}ITEMIDLIST;

typedef UNALIGNED ITEMIDLIST *LPITEMIDLIST;
typedef const UNALIGNED ITEMIDLIST *LPCITEMIDLIST;
```

Список представляет собой массив переменной длины элементов типа ITEMIDLIST. Но, в отличие от типа SHITEMID, в котором длина массива хранится в члене cb, количество элементов списка ITEMIDLIST нужно определять вручную. При этом анализируется признак завершения списка — наличие двух нулевых байтов. Поскольку переменная типа unsigned short также занимает два байта, то элемент списка с нулевым значением члена cb считается признаком конца списка.

Указатель на элемент списка ITEMIDLIST принято обозначать аббревиатурой PIDL. Как и пути, указатели PIDL существуют в двух вариантах: относительные и абсолютные. Первый, как правило, является относительным по отношению к "родительской" папке, а последний — отно-

сительным по отношению к папке рабочего стола. Поскольку папка рабочего стола является корневой в именованном пространстве оболочки, указатели PIDL, заданные по отношению к этой папке, часто называют *абсолютными* или *полностью определенными* (*fully qualified*). Указатели PIDL, заданные по отношению к своей родительской папке, называются *одноуровневыми* (*single-level*). Естественно, что объект именованного пространства, родительской папкой которого является папка рабочего стола, имеет PIDL, который можно считать одновременно и одноуровневым, и полностью определенным.

Для информации об идентификаторе объекта нужно выделять память точно так же, как и для строк символов. Как правило, память выделяется специальным компонентом программной оболочки, а возвращать эту память в операционную систему должен разработчик. Точно так же, как при работе со строками символов для возвращения памяти, ранее выделенной оператором new[], используется оператор delete[], нужно обратиться к компоненту выделения памяти программной оболочки, передать ему PIDL и потребовать вернуть память операционной системе. Указатель на COM-интерфейс выделения памяти IMalloc можно получить с помощью API-метода SHGetMalloc(). Возвращение памяти выполняется методом IMalloc::Free(), а при выделении памяти компонент оболочки обращается к методу IMalloc::Alloc(). Учтите, что для освобождения памяти, выделенной под идентификатор объекта именованного пространства, нельзя использовать оператор delete, функции free() или GlobalFree(). Поскольку большинство API-функций оболочек, использующих указатели PIDL, требует, чтобы вызывающая программа самостоятельно освобождала память, имеет смысл создать для этого специальную функцию (текст такой функции FreePidl() содержится в файле pidlhelp.cpp в папке Chapter 22 на прилагаемом к книге компакт-диске).

Извлечение PIDL объекта

Существует несколько стандартных способов извлечения PIDL объекта именованного пространства. Наиболее простой способ предполагает использование API-функции SHGetSpecialFolderLocation(). Как следует из имени этой функции, она возвращает абсолютный указатель PIDL на определенный объект папки, причем вызывающая программа должна затем освободить память, выделенную для этого объекта. Учтите, что функция SHGetSpecialFolderLocation() имеет довольно ограниченное применение. Ее можно использовать только для извлечения указателей на заданный набор специальных папок. Папка из этого набора задается аргументом nFolder этой функции, а результат — указатель PIDL на заданную папку — возвращается через аргумент ppidl. Например, для извлечения указателя PIDL на объект ***Control Panel*** нужно указать в качестве аргумента nFolder константу CSIDL_CONTROLS.

Другой, более сложный способ — использовать API-функцию SHBrowseForFolder(), которая выводит на экран диалоговое окно и дает возможность пользователю указать в нем объект, указатель на который требуется извлечь. Единственным аргументом функции SHBrowseForFolder() является адрес переменной структурного типа BROWSEINFO, члены которой задают параметры настройки выводимого диалогового окна. Функция возвращает абсолютный указатель PIDL выбранного пользователем объекта, причем вызывающая программа должна возвратить операционной системе память, выделенную для идентификатора объекта. Существуют определенные ограничения и в возможностях функции SHBrowseForFolder(). Они связаны с необходимостью вмешательства пользователя в процесс извлечения указателя. Более подходящий способ — использовать методы COM-интерфейса IShellFolder.

В структуре интерфейса IShellFolder имеется несколько весьма полезных методов работы с именованным пространством оболочки. Указатель на интерфейс извлекается API-функцией SHGetDesktopFolder(). Сразу после извлечения интерфейс связан с корневой папкой именованного пространства. Метод IShellFolder::ParseDisplayName() позволяет извлечь PIDL определенного объекта файловой системы. Поскольку к этому методу приходится обращаться довольно часто, имеет смысл создать для этого специальную функцию.

```
LPITEMIDLIST PIDLFromPath(AnsiString APath)
{
   LPITEMIDLIST lpidl = NULL;
   LPSHELLFOLDER lpsf;
   if (SUCCEEDED(SHGetDesktopFolder(&lpsf)))
   {
      wchar_t WPath [MAX_PATH];
      StringToWideChar(APath, WPath, MAX_PATH);
      unsigned long eaten;
      lpsf->ParseDisplayName(NULL, NULL, WPath,
                        &eaten, &lpidl, NULL);
      lpsf->Release();
   }
   return lpidl;
}
```

Функция ParseDisplayName() позволяет извлечь PIDL определенного объекта, заданного его именем, и не требует, в отличие от SHBrowseForFolder(), вмешательства пользователя в этот процесс. В структуре интерфейса IShellFolder имеются и другие методы манипулирования объектами именованного пространства, к которым мы будем неоднократно обращаться в дальнейших разделах этой главы.

Извлечение содержимого папки

Для сравнения двух элементов списка объектов при сортировке используется метод IShellFolder::CompareIDs(). Другой часто используемый метод — IShellFolder::GetDisplayNameOf() — позволяет извлечь отображаемое на экране имя объекта (файла или папки) по заданному указателю PIDL. Помимо этих вспомогательных методов, в структуре интерфейса IShellFolder имеются методы перечисления объектов EnumObjects() и связывания BindToObject(), которые играют ключевую роль при извлечении содержимого отдельной папки.

Прежде чем описывать методику использования метода IShellFolder::EnumObjects(), хочу остановить ваше внимание на том, почему этому методу отводится столь важная роль. Конечно, существуют достаточно апробированные способы извлечения содержимого каталога файловой системы, которые, в частности, предполагают использование VCL-функций FindFirst() и FindNext(). Но, как я уже не раз подчеркивал, эти и подобные им методы годятся только для работы с "физическими" объектами файловой системы, а для работы с виртуальными объектами именованного пространства их использовать нельзя. Именно по этой причине в программной оболочке используются указатели PIDL, которые работают с виртуальными объектами точно так же, как и с физическими. Если попытаться "пройтись" по содержимому папки рабочего стола с помощью функций FindFirst() и FindNext(), то при встрече с виртуальными объектами My Computer, Network Neighborhood и Recycle Bin про-

цесс прервется. Это ограничение относится к диалоговым окнам Open As и Save As, разработанным еще до изобретения программной оболочки Windows.

Интерфейс IShellFolder "в одиночку" не позволит перечислить содержимое отдельной папки — ему "на помощь" должен прийти COM-интерфейс IEnumIDList. Последний предназначен для перечисления элементов списка идентификаторов объектов. Метод IShellFolder::EnumObjects() формирует указатель на объект интерфейс IEnumIDList, метод Next() которого и выполняет переход к очередному объекту списка. Но, в отличие от функций FindFirst() и FindNext(), метод IEnumIDList::Next() извлекает относительный указатель PIDL, т.е. его можно использовать при работе и с объектами физической файловой системы, и с виртуальными объектами.

Идентификаторы объектов, описанные выше, несут информацию, "понятную" только программной оболочке, а для конечного пользователя такой идентификатор — не более чем набор произвольных цифр. Представьте себе, что произойдет, если вывести пользователю на экран список файлов каталога, состоящий из одних чисел. Программная оболочка связывает с каждым таким идентификатором "отображаемое" имя объекта. Например, имя Network Neighborhood связывается с виртуальной папкой, которая представляет верхний уровень установленного сетевого окружения. С каждым объектом именованного пространства связывается не только отображаемое имя, но и другая информация, например указатель на пиктограмму, которая используется для представления объекта на экране.

Для извлечения информации, связанной с указателем PIDL на определенный объект, — имени объекта, пиктограммы, типа объекта, выполняемого типа (для файлов приложений) и атрибутов файла (для объектов файловой системы) — используется функция SHGetFileInfo(). Эта же функция может извлекать и дескриптор системного списка изображений вместе с индексами пиктограмм, связанных с объектами именованного пространства.

Функция SHGetFileInfo() принимает в качестве аргумента абсолютный указатель PIDL, как, впрочем, и большинство других API-функций программной оболочки. С относительными указателями работают только методы интерфейса IShellFolder, поскольку объект интерфейса напрямую связан с определенной папкой. Здесь есть прямая аналогия с использованием только имени файла при работе в текущем каталоге файловой системы, когда нет необходимости в использовании абсолютного пути к файлу. Точно так же, как API-функция SetCurrentDirectory() может быть использована для задания текущего каталога файловой системы, метод интерфейса IShellFolder::BindToObject() можно применить для связывания объекта интерфейса с определенным объектом папки именованного пространства. Но, поскольку функция SHGetFileInfo() не является методом интерфейса IShellFolder, нужно с помощью специальной процедуры преобразовать относительный указатель PIDL в абсолютный. Эта операция выполняется с помощью вспомогательной функции MergeIDLists() (листинг 22.4).

Листинг 22.4. Вспомогательная функция MergeIDLists()

```
LPITEMIDLIST MergeIDLists(LPMALLOC lpMalloc,
          LPITEMIDLIST lpidl1,
          LPITEMIDLIST lpidl2)
{
   unsigned int cb1, cb2;

   if (lpidl1)   cb1 = GetPIDLSize(lpidl1);
   if (cb1 != 0)  cb1 = cb1-2;
   if (lpidl2)   cb2 = GetPIDLSize(lpidl2);
```

```
   if (cb2 != 0)  cb2 = cb2-2;

   const unsigned int total_size=cb1+cb2+2;
   LPITEMIDLIST lpidlAbsolute =
        reinterpret_cast<LPITEMIDLIST>(
                    lpMalloc->Alloc(total_size));
   if (lpidlNew)
   {
      ZeroMemory(lpidlAbsolute, total_size);
      CopyMemory(lpidlAbsolute, lpidl1, cb1);
      CopyMemory(reinterpret_cast<LPBYTE>(lpidlAbsolute)+cb1,
                              lpidl2, cb2);
      FillMemory(reinterpret_cast<LPBYTE>(lpidlAbsolute)+cb1+cb2,
                               2, 0);
   }
   return lpidlAbsolute;
}
```

Сначала с помощью вспомогательной функции `GetPIDLSize()` извлекаются размеры родительского и дочернего указателей `PIDL`, причем из каждого размера вычитается длина признака завершения списка (она равна 2). После того как определен размер абсолютного указателя `PIDL`, для его хранения него можно выделить память с помощью метода `IMalloc::Alloc()`, а затем скопировать содержимое в выделенный участок памяти. Фактически этот алгоритм выполняет те же операции, что и функция `strcat()` по отношению к строкам символов.

Желательно досконально разобраться в методике использования интерфейсов `IShellFolder` и `IEnumIDList`, поскольку аналогичная методика применяется и при формировании расширения пространства имен. Реализацию представленной методики вы найдете в программных кодах компонента `TExpTreeView`, файлы которого находятся на прилагаемом к книге компакт-диске.

Перемещение объектов оболочки

Одна из наиболее интересных функций программной оболочки Windows — возможность перемещения объектов. Например, во многих приложениях пользователю предоставляется возможность открыть файл, перетащив его значок из окна Windows Explorer на поле окна приложения. Расширение именованных пространств приводит к дальнейшему развитию этого процесса. Фактически эта методика позволяет перемещать объекты точно так же, как это делается с помощью Explorer. Для конечного пользователя тем самым значительно упрощается процесс переноса объектов. Применяемая при этом технология является по сути частным случаем OLE-технологии “перетащить-и-опустить” (drag-and-drop). В этой главе мы ограничимся описанием перемещения объектов оболочки из Explorer.

Прежде чем приложение получит извещение о событии перетаскивания (*drag-and-drop*), оно должно зарегистрировать свое окно в качестве возможного целевого объекта при выполнении перетаскивания. Для этого используется функция `RegisterDragDrop()`. В качестве первого аргумента функции передается дескриптор регистрируемого окна. Второй аргумент — экземпляр объекта интерфейса `IDropTarget`, с помощью которого в приложение передается информация о статусе события перетаскивания.

Интерфейс IDropTarget

В структуру интерфейса IDropTarget, помимо стандартных методов, унаследованных от интерфейса IUnknown, входят методы DragEnter(), DragOver(), DragLeave() и Drop(). Эти методы отличаются и от метода TControl::OnDragOver(), и от обработчика событий OnDragDrop. В классе реализации интерфейса IDropTarget анализируются значения аргументов этих методов и эта информация передается в остальную часть приложения. Объявление такого класса представлено в листинге 22.5.

Листинг 22.5. Класс реализации интерфейса IDropTarget

```
//=====================================================
#ifndef DropTargetH
#define DropTargetH
#include <ole2.h>
#define WM_OLEDRAGBASE WM_USER +101
#define WM_OLEDRAGDROP WM_OLEDRAGBASE +1
#define WM_OLEDRAGENTER WM_OLEDRAGBASE +2
#define WM_OLEDRAGOVER WM_OLEDRAGBASE +3
#define WM_OLEDRAGLEAVE WM_OLEDRAGBASE +4
//-----------------------------------------------------
class TDropTarget :public IDropTarget
{
   private:
   unsigned long m_FReferences; // количество ссылок
   bool m_FAcceptFormat; // флаг разрешения перетаскивания
   HWND m_FTargetHwnd; // окно, которому нужно передать извещение
   POINTL m_FCursorPoint; // текущая позиция указателя мыши

   // Вспомогательная функция для манипулирования
   // параметром pdwEffect
   DWORD KeysToEffect(DWORD grfKeyState);

   protected:
   // Управляющие функции уведомления
   virtual void __fastcall DoNotifyDragDrop(HDROP HDrop);
   virtual void __fastcall DoNotifyDragEnter();
   virtual void __fastcall DoNotifyDragOver();
   virtual void __fastcall DoNotifyDragLeave();

   protected:
   // Методы, унаследованные от IUnknown
   STDMETHOD(QueryInterface)(REFIID riid,void**ppvObj);
   STDMETHOD_(ULONG,AddRef)();
   STDMETHOD_(ULONG,Release)();

   // Методы IDropTarget
   STDMETHOD(DragEnter)(LPDATAOBJECT pDataObj,
                                DWORD grfKeyState,
                                POINTL pt,LPDWORD pdwEffect);
   STDMETHOD(DragOver)(DWORD grfKeyState, POINTL pt,
```

```
                                LPDWORD pdwEffect);
   STDMETHOD(DragLeave)();
   STDMETHOD(Drop)(LPDATAOBJECT pDataObj, DWORD grfKeyState,
                                POINTL pt,LPDWORD pdwEffect);

   public:
   __fastcall TDropTarget(HWND AHTargetControl);
   __fastcall ~TDropTarget();
};
#endif
//=====================================================
```

Структура класса `TDropTarget` разработана таким образом, чтобы обеспечить максимально возможный уровень переносимости. Этот класс не имеет унаследованных связей с библиотекой VCL. Его конструктор, используемый по умолчанию, принимает единственный аргумент типа `HWND`. Таким образом, для взаимодействия с прочими компонентами приложения можно использовать API-функцию `SendMessage()` и специально разработанные сообщения. Все это выполняется виртуальными методами `DoNotifyDragEnter()`, `DoNotifyDragOver()`, `DoNotifyDragLeave()` и `DoNotifyDragDrop()`.

Метод `DragEnter()`

Особый интерес в описанном классе `TDropTarget` представляют методы `DragEnter()`, `DragOver()`, `DragLeave()` и `Drop()`. Метод `DragEnter()` вызывается, когда пользователь, перетаскивая объект, пересекает рамку окна, ранее зарегистрированного в качестве объекта перетаскивания. Функция принимает четыре аргумента: указатель на объект интерфейса `IDataObject`, переменную типа `DWORD`, которая несет информацию о текущем состоянии ключей процесса перетаскивания, переменную структурного типа `POINTL`, которая содержит экранные координаты курсора мыши, и указатель на переменную типа `DWORD`, которая содержит (а затем получает) информацию о результате операции "сбрасывания" объекта. Программный код этого метода представлен в листинг 22.6.

Листинг 22.6. Метод `DragEnter()`

```
STDMETHODIMP TDropTarget::DragEnter(LPDATAOBJECT pDataObj,
                                    DWORD grfKeyState,
                                    POINTL pt,LPDWORD pdwEffect)
{
   // Текущие координаты мыши
   m_FCursorPoint = pt;

   // Уведомление целевого объекта о событии
   DoNotifyDragEnter();

   // Инициализация структуры FORMATETC
   FORMATETC fmtetc;

   ZeroMemory(&fmtetc, sizeof(FORMATETC));
   fmtetc.cfFormat = CF_HDROP;
   fmtetc.dwAspect = DVASPECT_CONTENT;
```

```
   fmtetc.lindex = -1;
   fmtetc.tymed = TYMED_HGLOBAL;

   // Единственный воспринимаемый формат источника – CF_HDROP
   m_FAcceptFormat = (pDataObj->QueryGetData(&fmtetc) == NOERROR);

   // Передать результат
   *pdwEffect = KeysToEffect(grfKeyState);
   return NOERROR;
}
```

Метод `TDropTarget::DragEnter()`, как и прочие методы интерфейса `IDropTarget`, вызывается функцией `DoDragDrop()`. В реализации метода `DragEnter()` обрабатываются данные, полученные через четыре аргумента, а затем через аргумент `pdwEffect` вызывающая программа извещается о результате выполнения операции "сбрасывания" объекта.

Этот аргумент играет ту же роль, что и аргумент `Accept` обработчика события `OnDragOver()` из библиотеки VCL. В реализации этого метода сначала сохраняются координаты курсора мыши, а затем с помощью метода `DoNotifyDragEnter()` приложению посылается уведомление о событии `DragEnter`. Далее проверяется тип перетаскиваемых через окно данных, для чего используется метод `QueryGetData()` объекта `IdataObject`, указанного аргументом `pDataObj`. Это выполняется не совсем так, как проверка параметра `Source` в обработчике события `OnDragOver()`, который входит в состав библиотеки VCL. В данном случае проверяется, имеет ли перетаскиваемый объект формат `CF_HDROP`, и затем соответственно изменяется аргумент `pdwEffect`. Эта операция выполняется с помощью метода `KeysToEffect()` (листинг 22.7), который в процессе принятия решения использует член-переменную `m_FAcceptFormat`.

Листинг 22.7. Метод `KeysToEffect()`

```
DWORD TDropTarget::KeysToEffect(DWORD grfKeyState)
{
   if (m_FAcceptFormat)
   {
      // Клавиши-модификаторы CRTL+SHIFT
      if ((grfKeyState &MK_CONTROL)&&(grfKeyState &MK_SHIFT))
      {
         return DROPEFFECT_LINK;
      }
      // Клавиша-модификатор CTRL
      if (grfKeyState &MK_CONTROL)
      {
         return DROPEFFECT_COPY;
      }

      // Клавиши-модификаторы отсутствуют
      return DROPEFFECT_MOVE;
   }
   return DROPEFFECT_NONE;
}
```

Через аргумент pdwEffect объект-источник информируется о возможности объекта-приемника работать с данным форматом, причем в качестве "зримой" реакции на это событие можно изменить форму курсора. Если формат приемлем и нажата клавиша <CTRL>, то метод KeysToEffect() присваивает члену pdwEffect значение DROPEFFECT_COPY. В этом случае Explorer выводит рядом с изображением курсора мыши хорошо вам знакомый символ +. Фактически, только методы TDropTarget::DragEnter() и Drop() получают указатель на объект данных. Методы DragOver() и DragLeave() используются в основном для формирования уведомления.

Метод Drop()

Метод TDropTarget::Drop() вызывается, когда пользователь "сбрасывает" перетаскиваемый объект на поле окна (листинг 22.8). Аргументы этого метода те же, что и для метода DragEnter().

Листинг 22.8. Метод Drop()

```
STDMETHODIMP TDropTarget::Drop(LPDATAOBJECT pDataObj,
                  DWORD grfKeyState,
                  POINTL pt, LPDWORD pdwEffect)
{
   *pdwEffect =KeysToEffect(grfKeyState);
   if (m_FAcceptFormat)
   {
      // Инициализация членов структуры FORMATETC
      FORMATETC fmtetc;
      ZeroMemory(&fmtetc,sizeof(FORMATETC));
      fmtetc.cfFormat =CF_HDROP;
      fmtetc.dwAspect =DVASPECT_CONTENT;
      fmtetc.lindex =-1;
      fmtetc.tymed =TYMED_HGLOBAL;

      // Пользователь "сбросил" объект.
      STGMEDIUM medium;
      HRESULT HResult =pDataObj->GetData(&fmtetc,&medium);
      if (SUCCEEDED(HResult))
      {
         // Координаты курсора мыши
         m_FCursorPoint =pt;
         // Послать уведомление объекту управления.
         DoNotifyDragDrop(medium.hGlobal);
         // Освободить использованную память.
         ReleaseStgMedium(&medium);
      }
      else return HResult;
   }
   return NOERROR;
}
```

Вместо метода QueryGetData(), который использовался в методе DragEnter(), в этой функции используется метод GetData(). Поскольку формат данных уже проанализирован, а результат анализа хранится в переменной-члене m_FAcceptFormat, можно извлекать данные из объекта. Поскольку работа идет с объектом в формате CF_HDROP, интересующие нас данные — дескриптор структуры DROPFILES (типа HDROP). Если выполнение функции GetData() завершится успешно, дескриптор сохраняется в члене hGlobal переменной структурного типа STGMEDIUM. Затем этот дескриптор передается объекту окна посредством обращения к методу DoNotifyDragDrop(). Поскольку объект окна более не нуждается в глобальном дескрипторе памяти, можно вернуть ранее выделенные ресурсы, вызвав метод ReleaseStgMedium().

Обработка события Drop

Как объект-приемник будет реагировать на "сброс" в него перемещаемого объекта (сообщение WM_OLEDRAGDROP), зависит от типа данных в этом объекте. Мы рассматриваем только объекты формата CF_HDROP, и окно приложения-приемника получает в таком случае дескриптор структуры DROPFILES, и использует функцию DragQueryFile() для извлечения разнообразной информации о перемещаемом объекте. Формат CF_HDROP вполне подходит для объектов типа файла или папки. Но для того чтобы приложение работало без сбоев, подумайте и над тем, как обрабатывать объекты в форматах CFSTR_FILEDESCRIPTOR и CFSTR_FILECONTENTS.

В качестве примера использования класса TDropTarget можете воспользоваться исходными текстами программ компонента TExpTreeView, которые находятся на прилагаемом компакт-диске. Особое внимание обратите на функцию DoTransferFiles() (ее код находится в файле EXPTREEVIEW.H), которая использует функции DragQueryFile() и SHFileOperation() для фактического перемещения объектов файла или папки.

Резюме

Объекты Active Server Pages и Active Server Objects представляют собой альтернативу устаревшей технологии использования Web-серверных приложений на базе CGI или ISAPI. Среда C++Builder 5 располагает ограниченными средствами поддержки ASP-объектов, но можно использовать для работы с ними и компоненты PageProducer и TableProducer, предназначенные для WebBroker (и даже компонент MidasPageProducer). Это позволяет создавать в среде C++Builder 5 полноценные приложения на базе ASP.

Кроме того, в этой главе вы имели возможность изучить технологию создания и внедрения ActiveForm-приложений. Тему формирования цифровых подписей мы не рассматривали. Тем, кто хочет узнать об этом аспекте работы ActiveForm-приложений, я рекомендую заглянуть на Web-сервер Microsoft. Там вы найдете список фирм, которые получили от Microsoft разрешение на создание соответствующих компонентов. Одной из них является фирма Verisign, которая имеет собственный сервер по адресу http://www.verisign.com/.

В последнем разделе этой главы мы познакомили вас с эффективными методами использования API-функций программной оболочки Windows, в частности с методикой использования указателей на объекты PIDL. На прилагаемом компакт диске вы найдете библиотеку функций pidlhelp, которая поможет вам создавать самостоятельно программы для работы с объектами программной оболочки.

Более сложные методы работы в C++Builder

ЧАСТЬ

IV

ПРЕДСТАВЛЕНИЕ ДАННЫХ С ПОМОЩЬЮ C++ BUILDER

ИСПОЛЬЗОВАНИЕ ИНТЕРФЕЙСА WIN32 API

МЕТОДЫ ИСПОЛЬЗОВАНИЯ МУЛЬТИМЕДИА

ПОКОРЕНИЕ НОВЫХ ВЕРШИН КОМПЬЮТЕРНОЙ ГРАФИКИ С ПОМОЩЬЮ ИНТЕРФЕЙСОВ DIRECTX И OPENGL

Глава

23

Представление данных с помощью C++Builder

Джеп Крамон
Стефан Махо
Дэн Баттерфилд

Пожалуй, преимущества Windows перед старой системой DOS наиболее отчетливо проявляются, с точки зрения рядового пользователя, в таких областях, как печать, управление печатью и представление данных в форме отчетов и диаграмм. Существенно усовершенствованная система печати больше не заставляет пользователя указывать порт и драйвер — от него требуется лишь выбрать принтер и запустить процесс печати, — а расширенные графические средства в сочетании с улучшенными параметрами разрешения экрана и принтера предоставляют более широкие возможности для высококачественного представления данных.

Относительно простая в работе система печати Windows расширяет поле деятельности программиста, однако даже с помощью функций API и VCL добавление возможностей профессиональной печати и представления данных может оказаться довольно непростой задачей. В этой главе рассматриваются основные и более сложные методы печати (на основе функций API и VCL), средства создания отчетов (с помощью компонентов пакета QuickReport компании QBS Software) и построения диаграмм (с использованием компонентов пакета TeeChart компании Steema SL). Пакеты QuickReport и TeeChart поставляются вместе с C++Builder.

Представление данных в отчетах

В этом разделе описано, как "выжать" максимум пользы из средства создания отчетов. Поскольку дизайн отчета сам по себе очень важен, то пользователи получают возможность его просмотра, сохранения и совместного доступа к нему без вывода на принтер.

О важности создания отчетов

Разработка баз данных, как правило, не обходится без этапа формирования отчетов. На создание нетривиально организованного отчета (на базе двух и больше таблиц) всегда уходило немало времени и сил. Но это как раз тот случай, когда игра стоит свеч, поскольку именно в отчетах подводятся итоги и приводятся результаты анализа данных. Ведь очень часто важны не сами данные, а их интерпретация.

К сожалению, к отчетам зачастую относятся как к простому инструменту вывода на печать содержимого базы данных, не используя при этом ни могучих возможностей компьютера, ни потенциала безбумажной технологии. А ведь в отчетах данные можно представить в различных формах, облегчая их восприятие пользователем. Отчет можно использовать, например, для получения списка клиентов в алфавитном порядке, но ведь к этому можно добавить и сведения о суммах сделанных ими закупок, сгруппировав их к тому же по периферийным отделениям, не так ли? Теперь-то пользователь будет иметь полную картину! Он сможет лучше оценить ситуацию, поскольку его знания (благодаря отчету) более систематизированы и конкретизированы. При этом, заметьте, отчет еще не печатался — вы просто просматривали его на экране компьютера.

Более того, сгенерированные отчеты можно хранить на диске. Кажущийся тривиальным, этот факт имеет весьма полезные последствия: в результате целое приложение может больше и не потребоваться — вполне достаточно одного лишь итогового средства просмотра, которое можно сделать (благодаря сети) предметом общего достояния, передать по электронной почте или упаковать в целях экономии памяти или бумаги. Ай да мы (в смысле: какая рационализация)!

Использование компонентов QuickReport для создания отчетов

C++Builder укомплектован средством создания отчетов QuickReport, созданным компанией QBS Software Ltd. (`www.qbss.com`), которая ранее называлась QuSoft Ltd. Отчеты создаются с использованием модели RAD (Rapid Application Development — ускоренная разработка

приложений) и компонуются вместе с программой. Обеспечивая абсолютную гибкость, они могут быть довольно сложными и включать C++-код. Версия Professional включает редактор, позволяющий сохранять макеты отчетов в отдельных файлах и настраивать их пользователями. Этот редактор относительно прост, поэтому пользователям будет нетрудно создавать собственные отчеты. Однако и у этой "медали" есть обратная сторона: более сложные отчеты придется формировать в интегрированной среде разработки (IDE) и компоновать вместе с программой или создавать с помощью другого инструмента.

Как уже упоминалось, сгенерированный отчет можно сохранить на диске (не говоря уже о возможности просмотра на экране и выводе на принтер). Собственный формат файла QuickReport — это специальный вид метафайла. Другими словами, это готовое изображение, которое предназначено для просмотра определенным образом. Он содержит атрибуты форматирования и позволяет выполнять масштабирование. С другой стороны, отчеты, сохраненные в этом формате, можно просматривать только с помощью приложения, которое использует компоненты, предоставленные в этом пакете.

Если отчеты предназначаются для тех, кто не будет работать с вашим приложением, то вам пригодятся фильтры (входящие в пакет QuickReport), которые позволят сохранять отчеты в различных форматах. Стандартная версия пакета QuickReport, поставляемая с C++Builder, включает фильтры для текстовых (использующих в качестве разделителя запятую) и HTML-форматов. Версия Professional, которая продается отдельно, включает дополнительные фильтры для Microsoft Excel, Microsoft Word (формат RTF) и метафайлов Windows. Это — популярные форматы и некоторые из них не зависят от платформы.

Философия настраиваемого средства просмотра

У пользователя может возникнуть необходимость в просмотре отчета, не выводя его на принтер. Это особенно важно в тех случаях, когда сгенерированные отчеты являются "продолжением" основного приложения, предлагая другое или более полное представление данных. Более того, просмотр сохраненных отчетов предоставляет множество существенных преимуществ.

Чтобы до конца понимать, как построено рассматриваемое здесь средство просмотра, необходимо иметь базовое представление о пакете QuickReport, т.е. понимать, как создается отчет, выведенный из компонента `TQuickRep`. Полезно также знать, что представляет собой компонент `TQRPreview`.

Для получения более подробной информации об этих компонентах обратитесь, пожалуйста, к документации пакета QuickReport.

Класс просмотра QuickReport `TQuickReportViewer` построен на основе компонента `TQRPreview` (см. прилагаемый к книге компакт-диск). Его создание преследовало следующие цели.

- Обеспечить идеальную интеграцию с вызывающим приложением. Это достигается с помощью двух методов, предназначенных для отображения средства просмотра: в его собственной форме или путем встраивания в "оконный" элемент управления другой формы (например, панели или вкладки).
- Разрешить динамический просмотр сгенерированных отчетов (выведенных из компонента `TQuickRep`). Этот процесс часто называют *предварительным просмотром* (previewing), однако я нахожу этот термин не вполне подходящим, поскольку он предполагает, что конечной целью является печать (что не всегда соответствует действительности).
- Позволить загрузку и просмотр сохраненных изображений отчетов (QRP-файлов).

Реализация первой и третьей функций демонстрируется в проекте QRViewer. Это средство просмотра QRP-файла должно убедить вас в том, насколько легко класс TQuickReportViewer можно интегрировать в приложение путем встраивания в элемент управления TTabSheet (рис. 23.1).

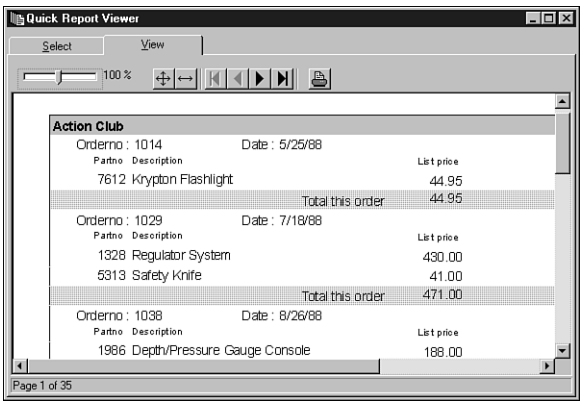


Рис. 23.1. Проект QRViewer демонстрирует, как встроить класс TQuickReportViewer в другую форму

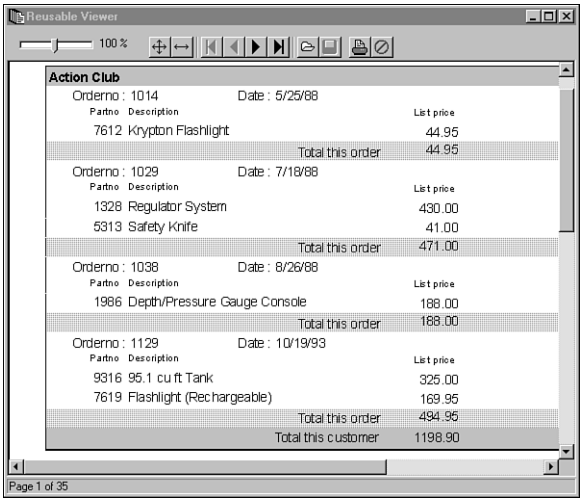


Рис. 23.2. Проект QRPreviewer демонстрирует класс TQuickReportViewer в его собственной форме

На заметку

Прилагаемый к книге компакт-диск содержит выполняемый файл QRViewer.exe, построенный на базе проекта QRViewer. Для простоты просмотра QRP-отчетов эту программу следует поместить в одну папку с ними. Она не использует никаких пакетов или библиотек динамической компоновки (RTL), т.е. является совершенно автономной. Она упакована (примерно на 50%), что упрощает ее пересылку по электронной почте вместе с сохраненными отчетами.

Реализация второй функции демонстрируется с помощью проекта QRPreviewer, который генерирует и позволяет просмотреть "пустой" отчет (он не содержит никаких данных — одно лишь название). В этом случае средство просмотра отображается в собственной форме и очень

напоминает стандартное окно предварительного просмотра. Но для предоставления дополнительных возможностей этот проект должен быть выведен из класса TQuickReportViewer. Кроме того, этот класс обеспечивает простой API-интерфейс, позволяющий просматривать и сохранять QRP-отчеты. Проект QRPreviewer в действии показан на рис. 23.2.

Обратите внимание на то, что класс TQuickReportViewer используется в обоих проектах и бесшовно интегрируется в программы.

Исходный текст программы средства просмотра

Настало время рассмотреть исходный текст программы настраиваемого средства просмотра. Все группы связанных методов предварительно проанализированы, а менее важные методы попросту опущены (полностью исходный код содержится в файле ReportViewer.cpp, который находится на прилагаемом к книге компакт-диске).

Ниже в полном объеме рассматривается класс TQuickReportViewer, хотя для просмотра сгенерированного отчета нужны только два его метода — Preview() и ClosePreview().

Бо*л*ьшая часть действий по инициализации членов этого класса выполняется в методе Init(), который можно вызвать из трех перегруженных конструкторов этого класса. В листинг 23.1 включен только один конструктор.

Листинг 23.1. Инициализация класса

```
__fastcall TQuickReportViewer::TQuickReportViewer( TComponent* Owner )
    : TForm(Owner),
      newQRPrinter_M(    NULL )
{
    Init( Owner, false ) ;
} // Конструктор


void __fastcall TQuickReportViewer::Init( TComponent* Owner,
                                          const bool showForm )
{
    // Следует ли передавать содержимое формы?
    // Блок if/else: "Оконный" элемент управления владельца также
    //      является новым родителем.
    // Примечание: Если владельцем является другая форма, это означает
    //             лишь, что они должны быть разрушены вместе.
    TWinControl* ownerAndParent = dynamic_cast<TWinControl*>( Owner );
    if( ownerAndParent && (!dynamic_cast<TForm*>(ownerAndParent)) )
        Client_Panel->Parent  = ownerAndParent ;
    else
    {
        Visible              = showForm ;
        Exit->Visible        = true ;
        ToolButton4->Visible = true ;
    } // В противном случае эта форма используется независимо.

    // Регистрируем компоненты Filter.
    SaveDialog->Filter = QRExportFilterLibrary->SaveDialogFilterString ;

    // Устанавливаем вид по умолчанию.
    FitClick( NULL ) ;
} //Init()
```

Все управляющие элементы формы средства просмотра на самом деле являются дочерними объектами панели Client_Panel. Она незаметна на этой форме, поскольку выровнена в соответствии со значением alCLient свойства Align и не имеет ни границ, ни каких-либо других видимых свойств. Если эта форма будет встроена внутрь другой формы или "оконного" элемента управления, владелец этой формы также выполнит роль родителя панели Client_Panel. В результате панель Client_Panel и все ее дочерние элементы управления будут повторно отображены в элементе управления ownerAndParent. И это замечательно, поскольку тогда элементы управления не нужно специально отображать в форме, которой они принадлежат. Иначе говоря, панель Client_Panel принадлежит классу QuickReportViewer и будет разрушена вместе с ним, несмотря на то, что она отображается на другой форме.

После внесения в средство просмотра компонентов фильтрации продуктов экспорта они автоматически регистрируются с помощью библиотеки QuickReport, после чего уже не составляет труда присвоить их свойству SaveDialog->Filter.

Достаточно просто решается задача отображения номера страницы с помощью приведенного ниже метода, как показано в листинге 23.2. Его стоит включить в проект, поскольку он используется и для отображения счетчика общего числа страниц в отчете, значение которого неизвестно до полного завершения генерации отчета. Эту динамику имеет смысл показать, поскольку QuickReport отображает первую страницу сразу после ее генерации, продолжая в то же время создавать остальные страницы, поэтому общее количество страниц будет постоянно возрастать по мере генерации отчета.

Однако возможны ситуации, когда общее количество страниц в отчете должно быть известно до начала его отображения. В таких случаях отчет должен быть полностью сгенерирован до вывода на экран. Это осуществляется путем вызова метода Prepare().

Листинг 23.2. Отображение номера страницы

```
void __fastcall TQuickReportViewer::DisplayPageNumber(
                                                    const int pageNumber ) const
{
    // Обновляем строку состояния.
    String page("Page " ) ;
    page += pageNumber ;

    int pageCount = QRPreview->QRPrinter->PageCount ;
    if( pageCount < 1 )
        pageCount = 1 ;
    page += " of " ;
    page += pageCount ;

    StatusBar->SimpleText = page ;

    // Обновляем кнопки.
    if( pageNumber <= 1 )
    {
        First->Enabled    = false ;
        Previous->Enabled = false ;
    } // Если это начало...
    else
    {
        First->Enabled    = true ;
```

```
        Previous->Enabled = true ;
    } // В противном случае - не начало.
if( pageNumber >= pageCount )
    {
        Next->Enabled = false ;
        Last->Enabled = false ;
    } // Если это конец...
    else
    {
        Next->Enabled = true ;
        Last->Enabled = true ;
    } // В противном случае - не конец.
} //DisplayPageNumber()
```

`Preview()` и `ClosePreview()` — единственные методы, необходимые для просмотра сгенерированного отчета. В начале метода `Preview()` переданному классу назначается обработчик событий `OnPreview()`. Он уведомляет QuickReport о необходимости использовать эту форму для отображения сгенерированного отчета путем совместного использования (со средством просмотра) канвы отчета. "Развитие событий" отображено в листинге 23.3.

Листинг 23.3. Просмотр сгенерированного отчета

```
void __fastcall TQuickReportViewer::Preview( TQuickRep* report )
// ОПИСАНИЕ: Это единственный метод, который необходимо
//           вызвать для просмотра сгенерированного отчета.
//
// ПРЕДОСТЕРЕЖЕНИЕ: код этой функции НЕ будет выполнен до
//    тех пор, пока не будет вызвана функция ClosePreview()
//    (если не будет определен макрос THREADSAFE_DBENGINE).
//
// ПРЕДОСТЕРЕЖЕНИЕ: Предполагается, что все наборы данных уже активны.
{
    assert( report ) ;

    // Новая канва QRPrinter всегда создается
    // методами Preview()/PreviewModeless().
    ClosePreview() ;

    // Назначаем обработчик событий, чтобы этот класс мог "увидеть" отчет.
    report->OnPreview = PrepareViewer ;

    // Форма может быть закрыта (скрыта).
    if( Client_Panel->Parent == this )
        Visible = true ;

    // Предварительный просмотр отчета.
    #ifdef THREADSAFE_DBENGINE
        TCursor Save_Cursor = Screen->Cursor;
        Screen->Cursor = crHourGlass;
        try
        {
```

```
                // ПРИМЕЧАНИЕ: PreviewModeless() возвращает
                //             управление после окончания генерации отчета.
          report->PreviewModeless();

          DisplayPageNumber( QRPreview->PageNumber ) ;
        } //try
        __finally
        {
            Screen->Cursor = Save_Cursor ;
        } //finally
    #else
            // ПРИМЕЧАНИЕ: Preview() НЕ возвращает управление,
            //          пока не будет вызван метод ClosePreview().
            //          Поэтому ВЕСЬ код после вызова Preview()
            //          НЕ будет выполнен до тех пор, пока
            //          не будет закрыто средство предварительного
            //          просмотра. По этой причине я не могу
            //          придать курсору вид песочных часов (курсор
            //          не восстановится даже после генерации).
        report->Preview();
    #endif

    // Удаляем обработчик событий, поскольку нет гарантии, что этот
    // экземпляр класса будет существовать при следующем просмотре
    // отчета. ПРИМЕЧАНИЕ: Этот код не зависит от времени.
    report->OnPreview = NULL ;
} //Preview()

void __fastcall TQuickReportViewer::PrepareViewer( TObject *sender )
{
    // ПРЕДОСТЕРЕЖЕНИЕ: при загрузке упакованных отчетов возникает
    //                  ошибка, но кнопка не станет недоступной,
    //                  поскольку она используется и для экспортирования.
    DefaultQRViewer_G->SaveReport->Enabled = true ;
    DefaultQRViewer_G->FitWidth->Down      = true ;

    // Используем канву отчета совместно со средством
    // предварительного просмотра.
    TQRPrinter* qrPrinter = dynamic_cast<TQRPrinter*>( sender ) ;
    assert( qrPrinter ) ;

    QRPreview->QRPrinter = qrPrinter ;
} //PrepareViewer()

void __fastcall TQuickReportViewer::ClosePreview( void )
// Освобождаем канву отчета.
// Вызываем этот метод, когда отчет больше не нужен.
// ПРИМЕЧАНИЕ: См. Preview() для других действий этого метода.
{
    if( QRPreview->QRPrinter )
    {
```

```
        QRPreview->QRPrinter->ClosePreview( QRPreview ) ;
        QRPreview->QRPrinter = NULL ;
    } //if

    SaveReport->Enabled  = false ;
} //ClosePreview()
```

Затем метод `Preview()` генерирует и отображает отчет. Предпочтительный метод — определить макрос `THREADSAFE_DBENGINE`, поскольку класс отчета становится в этом случае немодальным. К сожалению, этот вариант является единственно доступным, если используемый процессор баз данных гарантирует безопасность функционирования потоков (т.е. защиту от действий других потоков). В противном случае класс отчета ведет себя подобно модальной форме, т.е. он не возвращает управление источнику вызова до тех пор, пока отчет не будет закрыт нормальным обращением к методу `ClosePreview()`. Для использования макроса `THREADSAFE_DBENGINE` достаточно удалить символы комментария в его определении в начале файла `ReportViewer.cpp`.

Обратите внимание на то, что код в конце метода `Preview()` не зависит от времени.

Эти два метода позволяют сохранить отчет либо с помощью диалогового окна, либо без участия пользователя. С точки зрения вызывающего приложения, сохранение отчета — пара пустяков. И в самом деле, ему достаточно для этого вызвать метод `Save()` (приведенный в листинге 23.4) с двумя простыми параметрами. Не слишком миниатюрные размеры этого метода — результат выполнения проверки наличия ошибок и создания условий для экспортирования отчета в различных форматах файлов.

Этот метод, по сути, разделен на две части. При использовании формата QuickReport просто вызывается собственный метод `Save()` канвы принтера. Если же необходимо экспортирование, то сначала из библиотеки фильтров экспорта считывается информация о нужном типе и динамически создается соответствующий класс фильтра, который затем используется для генерации файла отчета.

Листинг 23.4. Сохранение сгенерированного отчета

```
void __fastcall TQuickReportViewer::SaveReportClick( TObject *Sender )
// Сохраняем или экспортируем изображение отчета.
// Примечание: если эта функция вызывается из другого модуля,
//             нужно на основании имени файла SaveDialog->FileName
//             узнать, был ли отчет сохранен (а не отменен).
{
    SaveDialog->FileName = "" ;

    if( SaveDialog->Execute() )
        Save( SaveDialog->FileName, SaveDialog->FilterIndex - 1 ) ;
    else
        SaveDialog->FileName = "" ;
} //SaveReportClick()


void __fastcall TQuickReportViewer::Save( String fileName, int filterIndex )
// 'fileName' должно включать корректное расширение.
//    Если оно отсутствует, добавляется соответствующий диалог.
// 'filterIndex' означает используемый формат файла.
//    Для собственного формата QuickReport (QRP)
```

```
//    используется значение 0. Любое другое значение
//    применяется как индекс для входа в библиотеку фильтров.
//    Эта библиотека заполнена компонентами фильтров,
//    помещенных в форму (с учетом порядка их создания).
{
    if( ! QRPreview->QRPrinter )
        Beep() ;

    else
    {
        assert( ! fileName.IsEmpty() ) ;

        // Считываем расширение при имени файла.
        int    len = fileName.Length() ;
        int    pos = fileName.Pos(".") ;
        String fileExt( fileName.SubString( pos + 1, len - pos ).UpperCase() ) ;

        // Сохраняем отчет (в собственном формате).
        if( filterIndex == 0 )
        {
            // Проверяем, соответствует ли расширение
            // выбранному фильтру.
            if( fileExt != "QRP" )
                Application->MessageBox(
                                "The file extension given does not "
                                "match the format selected.",
                                "Wrong file extension", MB_OK ) ;
            else
                QRPreview->QRPrinter->Save( fileName ) ;
        } //if для сохранения

        // экспорт
        else
        {
            // Считываем данные о фильтре.
            // Примечание: вычитаем 1 из filterIndex,
            //          поскольку в библиотеке фильтров
            //          начальный индекс равен 0.
            --filterIndex ;
            assert( (filterIndex >= 0) &&
                    (filterIndex < QRExportFilterLibrary->Filters->Count) ) ;
            TQRExportFilterLibraryEntry* filterEntry(
                                static_cast<TQRExportFilterLibraryEntry*>
                                ( QRExportFilterLibrary->
                                Filters->Items[filterIndex] ) ) ;

            // Проверяем, соответствует ли расширение
            // выбранному фильтру.
            if( fileExt != filterEntry->Extension )
                Application->MessageBox(
```

```
                                "The file extension given does not "
                                "match the format selected. ",
                                "Wrong file extension", MB_OK ) ;
            else
            {
                TQRExportFilter* exportFilter = NULL ;

                if( fileExt == "TXT" )
                    exportFilter = new TQRAsciiExportFilter( fileName ) ;
                else if( fileExt == "HTM" )
                    exportFilter = new TQRHTMLDocumentFilter( fileName ) ;
                else
                    exportFilter = new TQRCommaSeparatedFilter( fileName ) ;

                try
                {
                    assert( exportFilter ) ;
                    QRPreview->QRPrinter->ExportToFilter( exportFilter ) ;
                } //try
                __finally
                {
                    delete exportFilter ;
                } //__finally
            } //else - расширение имени файла допустимо
        } //else - экспорт
    } //else - сохраняемый отчет существует
} //Save()
```

И в этом случае для загрузки отчета используется один из двух вариантов (соответственно, и методов): с помощью диалогового окна либо без участия пользователя. Большая часть работы связана с подготовкой формы. Реальная загрузка осуществляется посредством класса `TQRPrinter`. Рассмотрим листинг 23.5.

Позволяя как сохранение, так и экспортирование, средство просмотра отчетов может загрузить файлы, сохраненные только в собственном формате QuickReport (`.qrp`). Для других файловых форматов необходимо использовать соответствующие приложения. Обратите также внимание на то, что `.qrp`-файлы представляют собой сохраненные, заполненные изображения отчетов, а не формы редактора отчета.

Листинг 23.5. Загрузка предварительно сохраненного отчета в формате QuickReport

```
void __fastcall TQuickReportViewer::LoadReportClick( TObject *Sender )
// Открываем QRP-файл.
// Примечание: если эта функция вызывается из другого модуля,
//          нужно на основании имени файла SaveDialog->FileName
//          узнать, был ли отчет открыт (а не отменен).
{
    OpenDialog->FileName = "" ;

    if( OpenDialog->Execute() )
        Load( OpenDialog->FileName ) ;
```

```
    else
        OpenDialog->FileName = "" ;
} //LoadReportClick()


void __fastcall TQuickReportViewer::Load( String fileName )
// Открываем QRP-файл.
// Примечание: QRP-файлы представляют собой сохраненные,
//          заполненные изображения отчетов, а не формы редактора отчета.
{
    TCursor Save_Cursor = Screen->Cursor;
    Screen->Cursor = crHourGlass;
    try
    {
        // Если нужно, назначаем новую канву QRPrinter.
        if( newQRPrinter_M == NULL )
            newQRPrinter_M = new TQRPrinter ;
        QRPreview->QRPrinter = newQRPrinter_M ;

        // Загружаем указанный файл.
        QRPreview->QRPrinter->Load( fileName ) ;
        QRPreview->PageNumber = 1 ;
        QRPreview->PreviewImage->PageNumber = 1 ;

        SaveReport->Enabled  = false ;
        DisplayPageNumber( QRPreview->PageNumber ) ;
        FitClick( FitWidth ) ;

        // Форма может быть закрыта (скрыта).
        if( Client_Panel->Parent == this )
            Visible = true ;
    } //try
    __finally
    {
Screen->Cursor = Save_Cursor ;
    } //finally
} //Load()
```

Резюме о настраиваемом средстве просмотра QuickReport

Итак, мы рассмотрели, как конструктор может встроить средство предварительного просмотра в другую форму, как сориентировать пакет QuickReport для использования настраиваемого средства просмотра, как сохранить и экспортировать сгенерированный отчет, а также как загрузить ранее сохраненный отчет. Неудивительно, что все перечисленные возможности соответствуют открытому (`public`) интерфейсу класса `TQuickReportViewer`.

Цель использования настраиваемого средства просмотра QuickReport — улучшить интеграцию в одном приложении. Представленный здесь класс можно использовать либо в собственной форме, либо путем встраивания в другую форму или “оконный” элемент управления.

Описанное средство просмотра можно использовать для отображения сгенерированных отчетов, которые могут быть сохранены во многих популярных форматах файлов. Это позволяет без проблем передавать их другим пользователям или архивировать. То же средство просмотра можно затем использовать для отображения отчетов, сохраненных в собственном формате QuickReport.

Отчеты — важная часть приложений, работающих с базами данных. Поэтому столь существенна их адекватная интеграция.

Печать текста и графических изображений

В этом разделе рассматриваются различные методы печати текста и графических изображений. Вы познакомитесь с методами непосредственного вывода данных и текста на принтер без применения библиотечной инкапсуляции *интерфейса с графическими устройствами* (Graphics Device Interface — GDI), а также методами печати с использованием *библиотеки визуальных компонентов* (Visual Component Library — VCL).

Предполагается, что читатель обладает базовыми знаниями о том, как работает класс `TPrinter`. Если вам до сих пор не приходилось использовать этот класс, я рекомендую рассмотреть пример `Printing`, который находится в подпапке `Examples\Apps\Printing` папки `C++Builder`.

Печать текста

В этом разделе описываются различные аспекты печати текста, начиная с вывода неформатированных данных прямо на принтер (несмотря на существование замечательной системы печати, особенно GDI-интерфейса) и заканчивая печатью превосходно отформатированного текста с использованием шрифтов различных начертаний и размеров. Более подробно я остановлюсь на возможностях непосредственной печати текста и использовании класса `TCanvas`. Изменение размера шрифта и прочих атрибутов форматирования выводимого на принтер текста — это просто расширение функций, описанных в данном разделе.

Непосредственный вывод текста и данных на принтер

Прежде всего рассмотрим, как вывести текст и данные прямо на принтер, не используя класс `TCanvas`, который олицетворяет VCL-инкапсуляцию GDI-интерфейса.

Самый простой и в то же время достаточно проблемный для большинства пользователей способ — пересылка текста и данных прямо на принтер. Используя несколько простых функций Win32 API, мы можем справиться с этой задачей в один миг.

Под управлением Windows 3.x мы могли для этого использовать такие функции, как `Escape()` или `EscapeEx()`, но для среды Win32 эти API-функции уже устарели и поэтому не будут здесь рассматриваться ввиду существования более приемлемых альтернатив.

Наилучший способ прямой записи на принтер под управлением Win32 — использовать функцию `WritePrinter()`. Эта функция информирует систему буферизации принтера о предстоящей прямой записи на принтер, заданный в вызове функции. Листинг 23.6 содержит пример настройки принтера на использование функции `WritePrinter()`. Ключевыми словами в данном контексте являются `OpenPrinter()`, `WritePrinter()`, `ClosePrinter()`, `StartDocPrinter()`, `StartPagePrinter()`, `EndPagePrinter()`, `EndDocPrinter()`, `DOC_INFO_1`, `PRINTER_DEFAULTS` и `HANDLE`.

Листинг 23.6. Прямой вывод на принтер с помощью функции `WritePrinter()`

```
// -- Полностью исходный код приведен в проекте PrintSample1.
void __fastcall TMainForm::Print(char *szPrinterName)
{
    HANDLE hPrinter;      // Дескриптор принтера.
    PRINTER_DEFAULTS pd;  // Права доступа к принтеру только для WinNT/Win2K,
                          // системой Win9x игнорируется.

    // Инициализируем стандартные параметры принтера.
    ZeroMemory(&pd, sizeof(pd));
    // Права доступа в Windows NT.
    pd.DesiredAccess = PRINTER_ACCESS_USE;

    // Открываем принтер и получаем для него дескриптор,
    // который хранится в переменной hPrinter.
    if(OpenPrinter(szPrinterName, &hPrinter, &pd))
    {
        // Используем структуру DocInfo1.
        DOC_INFO_1 docInfo1;
        // Инициализируем структуру DocInfo1.
        ZeroMemory(&docInfo1, sizeof(docInfo1));

        // --- Устанавливаем информацию Doc1. ---
        // Имя документа, которое приводится в окне программы
        // управления печатью (Print Manager).
        docInfo1.pDocName       = "Print Sample 1";

        // Порт/файл (здесь не используется)
        docInfo1.pOutputFile    = 0;

        // Данные посылаются в RAW-формате (Форматы RAW
        // и EMF поддерживаются как Win9x, так и
        // Win NT). Win NT также поддерживает TEXT.
        docInfo1.pDatatype      = "RAW";


        // --- Начало документа (второй параметр задает
        // тип Doc Info) ---
        if(!StartDocPrinter(hPrinter, 1,  (LPBYTE)&docInfo1))
        {
            // Что-то неисправно (для выяснения причины
            // неисправности используйте функцию GetLastError()).
            return;
        }

        // --- Сообщаем принтеру о нашем намерении печатать.
        if (!StartPagePrinter(hPrinter))
        {
            // Что-то неисправно (для выяснения причины
            // неисправности используйте функцию GetLastError()).
            return;
```

```
        }


        // ---------------------------------
        // --- Начало кода WritePrinter ---
        // ---------------------------------
        DWORD dwWritten;     // Длина реально посланных данных.
        AnsiString asBuffer; // Буфер для хранения отправляемых данных/текста.

        // Записываем на принтер "Привет, мир!" (включая символы
        // возврата каретки и перевода строки (новой строки)).
        asBuffer = "Привет, мир!\ r\ n";
        if(!WritePrinter(hPrinter, (LPVOID)asBuffer.c_str(),
                         asBuffer.Length(), &dwWritten))
        {
             // Ошибка, возникшая при отправке данных на принтер.
             // Для выяснения причины неисправности используйте LastError.
             return;
        }

        // Записываем на принтер символ перевода
        // страницы (он также называется символом новой страницы)
        asBuffer = "\ f";
        if(!WritePrinter(hPrinter, (LPVOID)asBuffer.c_str(),
                         asBuffer.Length(), &dwWritten))
        {
             // Ошибка, возникшая при отправке данных на принтер.
             // Для выяснения причины неисправности используйте LastError.
             return;
        }

        // ---------------------------------
        // --- Конец кода WritePrinter ---
        // ---------------------------------

        // Сообщаем принтеру о завершении печати.
        if (!EndPagePrinter(hPrinter))
            return;

        // Конец документа.
        if(!EndDocPrinter(hPrinter))
            return;

        // Закрываем принтер.
        ClosePrinter(hPrinter);
    }
}
```

Несмотря на то что мы используем только несколько функций, для печати считанных строк текста приходится настраивать принтер, используя довольно значительный объем программного кода. Чтобы облегчить эту задачу, я создал класс `TPrintRaw`, который инкапсули-

рует API-интерфейс принтера с помощью нескольких простых методов. В листинге 23.7 приводится пример использования класса TPrintRaw, причем выполняемые им действия аналогичны действиям примера из листинга 23.6. Исходный код класса TPrintRaw можно найти на прилагаемом к книге компакт-диске.

Листинг 23.7. Прямой вывод на принтер с помощью класса TPrintRaw

```
// -- Полностью исходный код приведен в проекте PrintSample2.
void __fastcall TMainForm::Print(char *szPrinterName)
{
    // Создаем новый экземпляр класса TPrintRaw.
    TPrintRaw *pPrintRaw = new TPrintRaw();
// Записываем заголовок задания печати в системе управления принтером.
    pPrintRaw->Title = "Print Sample 2";

    // Записываем имя принтера, который собираемся использовать.
    pPrintRaw->PrinterName = szPrinterName;

    // Подготовка к печати.
    pPrintRaw->BeginDoc();

    // Печать неформатированных данных (возврат кода 0 означает
    // успешное выполнение).
    // Записываем на принтер "Привет, мир!" (включая символы
    // возврата каретки и перевода строки (новой строки)).
    AnsiString asBuffer = "Привет, мир!";
    if (pPrintRaw->WriteLine(asBuffer))
        Application->MessageBox("An error occurred while sending"
                                " data to the printer",
                                "Printer Sample 2", MB_OK |
                                MB_ICONERROR);

     // Записываем на принтер символ перевода
    // страницы (он также называется символом новой страницы).
    // Примечание: это заставляет принтер прогнать бумагу.
    pPrintRaw->NewPage();

    // Печать завершена.
    pPrintRaw->EndDoc();

    // Освобождение экземпляра.
    delete pPrintRaw;
}
```

Как видно из листинга, при нормальном выполнении задания печати с помощью библиотеки VCL сначала вызывается функция BeginDoc(), а затем выполняются различные задачи, связанные с печатью (например, WriteLine() и другие). Вывод текста на принтер необходимо завершить вызовом метода EndDoc().

В табл. 23.1 приведен список методов и свойств, доступных в классе TPrintRaw.

Таблица 23.1. Методы и свойства класса `TPrintRaw`

Имя	Тип	Описание
`BeginDoc()`	Метод	Дает команду системе буферизации принтера приступить к печати нового документа
`EndDoc()`	Метод	Информирует систему буферизации принтера о завершении печати документа
`NewLine()`	Метод	Вставляет в документ новую строку (перевод строки)
`NewPage()`	Метод	Вставляет в документ новую страницу (перевод страницы). Не забудьте вызвать этот метод после печати последней страницы, чтобы заставить принтер прогнать бумагу
`Write()`	Метод	Записывает в документ строку
`WriteBuffer()`	Метод	Записывает в документ данные или текст
`WriteLine()`	Метод	Записывает в документ строку и вставляет символ новой строки в конце записанного текста
`WriteFile()`	Метод	Выводит на принтер содержимое файла (двоичного или ASCII)
`PrinterName`	Свойство	Считывает или устанавливает имя используемого принтера
`Printing`	Свойство	Если в данный момент происходит печать, возвращает истинное значение; в противном случае — ложное
`Title`	Свойство	Считывает или устанавливает заголовок документа, отображаемого в окне программы управления печатью (Print Manager)

Вывод текста на принтер с использованием библиотеки VCL

Библиотека визуальных компонентов обеспечивает инкапсуляцию интерфейса Win32 GDI в виде класса `TCanvas`, к которому можно получить доступ посредством объекта `Printer()`. Класс `TCanvas` содержит ряд перечисленных ниже методов, предназначенных для управления принтером и записи текста.

- Метод `TextExtent()` возвращает пространственные размеры (ширину и высоту) строки, записанной текущим шрифтом канвы, в виде значений "классового" типа `TSize`. `TextExtent()` — это тестовая функция, которая не записывает никакого текста на канву и не обновляет свойство `PenPos`. То же справедливо и для методов `TextHeight()` и `TextWidth()`.
- Метод `TextHeight()` возвращает то же значение, что и `Printer()->Canvas->TextExtent("Привет, мир!").cy`.
- Метод `TextWidth()` возвращает то же значение, что и `Printer()->Canvas->TextExtent("Привет, мир!").cx`.
- Метод `TextOut()` записывает строку на канву, начиная с точки (x,y), и обновляет свойство `PenPos` до конца строки.
- Метод `TextRect()` записывает заданную строку, начиная с точки (x,y) канвы, внутри ограничительного прямоугольника. Этот прямоугольник ограничивает размер отображаемого текста. Любая часть текста, не попадающая в границы прямоугольника, отсекается и не отображается. Точка (x,y) принадлежит верхнему левому углу прямоугольной области, занимаемой отображаемым текстом.

Для просмотра параметров методов класса TCanvas нажмите клавишу <F1>.

Для управления позицией рисования и печати (позицией пера) на канве используйте свойство PenPos.

Все размеры приводятся в пикселях, а координаты верхнего левого угла листа бумаги соответствуют значению (0,0).

Ниже приведен небольшой пример использования метода TextRect().

```
TRect rect;
rect.Top = 30;
rect.Left = 30;
rect.Bottom = 110;
rect.Right = 110;
Canvas->TextRect(rect, 35, 35, "Привет, мир!");
```

При выполнении этого примера будет отображен указанный текст внутри прямоугольника с заданными координатами верхнего левого (30,30) и нижнего правого (110,110) углов. Текст отображается, начиная с позиции (35,35), которая находится внутри ограничительного прямоугольника. Как именно будет отображен заданный текст — целиком или только частично — зависит от размеров занимаемой им области.

При этом необходимо отметить следующие два момента.

- К свойству Canvas класса TPrinter можно получить доступ только после вызова функции BeginDoc() для объекта Printer(). Это также означает, что обращения к функциям Win32, которым нужны дескрипторы (например, функция GetDeviceCaps()), до вызова функции BeginDoc() должны использовать выражение Printer()->Handle, а после ее вызова — выражение Printer()->Canvas->Handle.
- Не следует создавать отдельные экземпляры класса Tprinter путем вызова конструктора TPrinter. Вместо этого используйте глобальный экземпляр класса TPrinter, который возвращается методом Printer(). Чтобы воспользоваться классом TPrinter, необходимо включить в свой проект заголовочный файл printers.hpp.

В листинге 23.8 приведен пример использования библиотеки VCL для перевода принтера в режим построчной печати за счет применения функций, обеспечивающих перенос слов на новую строку.

Листинг 23.8. Пример построчно печатающего принтера

```
// --- Полностью исходный код приведен в проекте PrintSample3.
// Выводим содержимое Memo-поля на выбранный принтер.
// Используемые здесь методы позволяют печатать двумя
// способами (с использованием переноса слова или без
// этого), которые выбираются с помощью элемента
// управления TCheckBox, именуемого WordWrapCheckBox.
void __fastcall TMainForm::Print()
{
    // --- Локальные переменные.--- //
    int iLineNumber=1;              // Номер печатаемой строки на
                                    // текущей странице.
```

```
AnsiString asCurrentLine;        // Текст текущей (печатаемой
                                 // в данный момент) строки.
int iStartX = 0, iStartY = 0;    // Отправная (X,Y) позиция печати.

// Установка заголовка печати.
Printer()->Title = "Print Sample 3";

// Запуск задания печати.
Printer()->BeginDoc();

// Обеспечиваем использование одинаковых шрифтов принтера и memo-поля.
Printer()->Canvas->Font->Assign(Memo->Font);

// Высота строки (вычисляется только один раз).
int iLineHeight = Printer()->Canvas->TextHeight("");

// --- Начало построчной записи. --- //
for(int iLine=0; iLine < Memo->Lines->Count; iLine++)
{
    // Читаем текущую строку, подлежащую печати.
    asCurrentLine = Memo->Lines->Strings[iLine];

    // Проверяем, не превышает ли строка, подлежащая печати, ширину
    // страницы. Если превышает, и пользователь выбрал печать
    // с переносом слов, то будет выполнен код, реализующий перенос
    // слов на новую строку; в противном случае
    // выполняется обычная печать без переноса слов.
    bool bExceedsWidth;
    bExceedsWidth = Printer()->Canvas->TextWidth(asCurrentLine)
                    +iStartX >= Printer()->PageWidth;

    // Перенос слов?
    if (WordWrapCheckBox->Checked && bExceedsWidth)
    {
        // -------------------------- //
        // --- Печать с переносом слов (WordWrap). --- //
        // -------------------------- //

        // Временная переменная.
        int iCount=1;

        // Начинаем копирование по одному символу с текущей
        // строки, подлежащей печати, во временную строку
        // до тех пор, пока не превысим ширину.
        Printer()->Canvas->PenPos = TPoint(iStartX,
                        Printer()->Canvas->PenPos.y);

        while(iCount <= asCurrentLine.Length())
        {
            // Выводим на принтер по одному символу.
```

```
        Printer()->Canvas->TextOut
                  (
                  Printer()->Canvas->PenPos.x,
                  iStartY+iLineHeight*iLineNumber-1,
                  asCurrentLine[iCount]
                  );

        // Если это не самый последний символ,
        // проверяем, не будет ли превышена ширина
        // страницы при печати следующего символа.
        if (iCount != asCurrentLine.Length())
        {
            if (Printer()->Canvas->PenPos.x +
                Printer()->Canvas->TextWidth
                         (
                         asCurrentLine[iCount+1]) >
                         Printer()->PageWidth
                         )
            {
                // Новая строка.
                iLineNumber++;
                Printer()->Canvas->PenPos = TPoint
                            (
                            iStartX,
                            Printer()->Canvas->PenPos.y+
                            iLineHeight
                            );
            }
        }

        // Новая страница?
        if (Printer()->Canvas->PenPos.y+iLineHeight >=
            Printer()->PageHeight)
        {
            // Начинаем новую страницу.
            Printer()->NewPage();
            iLineNumber = 1;
            Printer()->Canvas->PenPos = TPoint
                            (
                            iStartX,
                            Printer()->Canvas->PenPos.y
                            );
        }

        // Следующий символ.
        iCount++;
    }  // ! while

    // --- По завершении вывода строки с переносом слов
    //     обычно вставляется новая строка. При этом
```

```
    //      нужно проверить, не требуется ли также
    //      переход на новую страницу. ---

    // Новая строка.
    iLineNumber++;
    Printer()->Canvas->PenPos = TPoint
                          (
                          iStartX,
                          Printer()->Canvas->PenPos.y+
                          iLineHeight
                          );
    // Новая страница?
    if (Printer()->Canvas->PenPos.y+iLineHeight >=
        Printer()->PageHeight)
    {
        // Начинаем новую страницу.
        Printer()->NewPage();
        iLineNumber = 1;
        Printer()->Canvas->PenPos = TPoint
                          (
                          iStartX,
                          Printer()->Canvas->PenPos.y
                          );
    }
}
else
{
    // ------------------------------ //
    // --- Печать без переноса слов.--- //
    // ------------------------------ //

    // Используется -1, поскольку начальное значение
    // iLineNumber равно 1, а не 0, в противном случае
    // получим ошибку смещения на единицу.
    Printer()->Canvas->TextOut
                     (
                     iStartX,
                     iStartY+iLineHeight*iLineNumber-1,
                     asCurrentLine
                     );

    // Проверяем PenPos, чтобы узнать, не будет ли превышена высота
    // страницы при следующем выводе строки. Если - да, нужно
    // вставить новую страницу!
    // NB: умножаем на 2, поскольку значение PenPos.y
    // еще не обновлялось для ссылки на следующую строку.

    // Новая страница?
    if (Printer()->Canvas->PenPos.y+2*iLineHeight >=
        Printer()->PageHeight)
```

```
            {
                // Начинаем новую страницу.
                Printer()->NewPage();
                iLineNumber = 1;
            }
            else
            {
                iLineNumber++;       // Готовность к
                                     // следующей строке.
            }
        }  // ! if (перенос слов)
    }  // ! for (все строки)

    // Обновляем заголовок, содержащий количество напечатанных страниц.
    Caption = Application->Title + " - " +
                    Printer()->PageNumber + " Page(s) Printed";

    // Конец задания печати.
    Printer()->EndDoc();
}
```

Печать графических изображений

В этом разделе мы покажем, как правильно вывести на принтер растровые и другие типы изображений. У тех, кто пытается это сделать, возникает множество недоразумений, и мы постараемся показать правильный путь решения этой задачи и указать на недостатки в существующих методах.

Печать растровых изображений с использованием библиотеки VCL

Класс `TCanvas` содержит две функции для переноса (blitting) растров с одной канвы на другую: `Draw()` и `StretchDraw()` (под переносом понимается копирование битов изображения из одного контекста устройства в другой). Однако ни одна из этих функций должным образом не справляется с переносом на канву принтера, и на это есть ряд причин. Как упоминалось выше, класс `TCanvas` представляет собой инкапсуляцию Windows-интерфейса GDI и включает набор процедур, выполняющих операции в *контексте устройства* (Device Context — DC). Контекст устройства — это структура, которая определяет набор графических объектов, их атрибуты и графические режимы, оказывающие влияние на результат. Графическими объектами могут быть перья для проведения линий, кисти для закрашивания и заполнения (заливки), растры для копирования или прокрутки фрагментов изображения на экране, палитры для определения наборов доступных цветов, области для отсечения и других операций, а также контур для операций закрашивания и рисования.

Проблема использования функций `Draw()` и `StretchDraw()` состоит в том, что они работают со стандартными растрами типа DDB (device dependent bitmap — аппаратно-зависимый растр), которые зависят от типа драйвера устройства и даже от версии драйвера. Это означает, что растровое изображение типа DDB для экрана отличается от такого же DDB-растра для принтера. Это создает проблемы при переносе растров, поскольку DDB-растр не содержит заголовочной информации, необходимой для соответствующего переноса битов между различными устройствами. Для получения более подробной информации о переносе растровых изображений прочитайте статью “Blitting Between DCs for Different Devices Is Unsupported”

по адресу http://support.microsoft.com/support/kb/articles/q195/8/30.asp. Идентификационный номер этой статьи (Article ID) — Q195830.

DDB-растры используются довольно широко, поскольку они требуют меньше памяти и обрабатываются быстрее, чем аппаратно-независимые растры (device independent bitmap — DIB). Некоторые драйверы принтеров обнаруживают DDB-перенос и корректируют проблемы, которые могут возникнуть, но, во-первых, — не все, а, во-вторых, если и корректируют, то недостаточно хорошо. Отсюда вывод: вообще не следует использовать функции Draw() или StretchDraw() для канвы принтера.

Чтобы решить эту проблему, компания Microsoft разработала GDI-функции для преобразования DDB- в DIB-растры: GetDIBits() и StretchDIBits(). К сожалению, эти функции также не лишены проблем. А именно, при вызове функции GetDIBits() для чтения битов изображения палитра зачастую игнорируется. Лучше всего в этом случае создать действительно хороший DIB-заголовок и цветовую таблицу. Если изображение хранится в .res- или дисковом файле либо в виде других надежных DIB-ресурсов, задача решается просто: не нужно считывать DIB-растр непосредственно из источника и писать VCL-код для переноса изображения. Заметим, что создание хорошего средства чтения растра — не такая уж тривиальная задача (и VCL тому доказательство). Прочитать цветовую таблицу с экранного изображения — еще труднее, так как следует принять во внимание палитру, т.е. предусмотреть возможность того, что вы имеете дело с палитровым устройством, которое включает 1-, 4- и 8-битовые изображения, имеющие цветовые таблицы. Наконец, если принтер является палитровым устройством (что случается довольно редко), вам необходимо сделать палитру доступной для принтера. Под управлением Windows NT почти каждый принтер имеет по крайней мере структуру палитры (даже если она и не используется), но в данном случае это не так важно.

К счастью, в библиотеке VCL предусмотрены оболочки для функции GetDIBits(), именуемые GetDIBSizes() и GetDIB(), которые включают обработку палитр. Листинг 23.9 — пример использования этих оболочек.

Листинг 23.9. Печать с использованием функции StretchDIBits()

```
// --- Полностью исходный код приведен в файле
// StretchBltBitmap.cpp (в папке PrintTools). ---
// pCanvas -> канва приемника.
// ix, iY, iWidth и iHeight -> параметры определения
//                             прямоугольника на канве для
//                             переноса DDB-растра
//                             (Device Dependent Bitmap).
// pBitmap -> растр, который нужно перенести.
void __fastcall StretchBltBitmap(TCanvas *pCanvas,
                                 int iX, int iY,
                                 int iWidth, int iHeight,
                                 Graphics::TBitmap *pBitmap)
{
    // --- Локальные переменные. --- //
    unsigned int iInfoHeaderSize=0;  // Размер заголовка с информацией о растре.
    unsigned int iImageSize=0;       // Размер растрового изображения.
    BITMAPINFO *pBitmapInfoHeader;   // Указатель на заголовок
                                     // с информацией о растре.
    unsigned char *pBitmapImageBits; // Указатель на биты
                                     // растрового изображения.
```

```
    // Сначала мы вызываем функцию GetDIBSizes() для определения объема
    // памяти, которую нужно выделить до вызова функции GetDIB().
    // NB: GetDIBSizes() входит в состав VCL.
    GetDIBSizes(pBitmap->Handle,    // Дескриптор растра.
                iInfoHeaderSize,    // Размер заголовка с информацией
                                    // о растре (размер данных).
                iImageSize);        // Размер растрового изображения.

    // Затем выделяем память в соответствии с информацией,
    // которую возвратила функция GetDIBSizes().
    pBitmapInfoHeader = new BITMAPINFO[iInfoHeaderSize];
    pBitmapImageBits  = new unsigned char[iImageSize];

    // Вызываем GetDIB() для преобразования аппаратно-
    // зависимого растра в аппаратно-независимый растр (DIB).
    // NB: GetDIB() входит в состав VCL.
    GetDIB(pBitmap->Handle,     // Дескриптор растра.
           pBitmap->Palette,    // Палитра растра.
           pBitmapInfoHeader,   // Заголовок с информацией о растре (данные).
           pBitmapImageBits);   // Биты растра.


    // Наконец, скопируем цветовые данные для прямоугольника
    // пикселей в DIB (в заданный прямоугольник приемника).
    // NB: StrecthDIBits() поддерживается интерфейсом Win32 API.
    StretchDIBits(pCanvas->Handle,    // Канва приемника.
                  iX,                 // Координата X верх. угла приемника.
                  iY,                 // Координата Y верх. угла приемника.
                  iWidth,             // Ширина приемника.
                  iHeight,            // Высота приемника.
                  0,                  // Координата X верх. угла источника.
                  0,                  // Координата Y верх. угла источника.
                  pBitmap->Width,     // Ширина источника.
                  pBitmap->Height,    // Высота источника.
                  pBitmapImageBits,   // Биты изображения.
                  pBitmapInfoHeader,  // Данные изображения (заголовок с
                                      // информацией о растре).
                  DIB_RGB_COLORS,     // Применение: цветовая таблица содержит
                                      // литеральные RGB-значения.
                  SRCCOPY);           // Код операции: копирует прямоугольник
                                      // источника прямо в
                                      // прямоугольник приемника.

    // Очистка. Освобождаем выделенную память.
    delete []pBitmapInfoHeader;
    delete []pBitmapImageBits;
}
```

Листинг 23.10 — пример из проекта `PrintSample4` (он находится на прилагаемом к книге компакт-диске), который показывает, как вызвать эту функцию. Прежде чем открывать проект `PrintSample4`, необходимо инсталлировать компонент `TPrintDIB` в C++Builder.

Листинг 23.10. Печать с использованием функции StretchBltBitmap

```
// Полностью исходный код приведен в проекте PrintSample4.
void __fastcall TMainForm::PrintBmpVCLButtonClick(TObject *Sender)
{
    // Устанавливаем портретную ориентацию.
    Printer()->Orientation = poPortrait;

    // Устанавливаем заголовок для задания печати.
    Printer()->Title = "Print Sample 4 (VCL)";

    // Начало печати.
    Printer()->BeginDoc();

    // Определяем приемлемый коэффициент масштабирования (работает
    // корректно только для изображений, которые меньше размера
    // бумаги. Выходит, что масштабирования не происходит!).
    float fPrinterVert = (float)GetDeviceCaps
                               (
                               Printer()->Canvas->Handle,
                               LOGPIXELSY
                               );
    float fPrinterHorz = (float)GetDeviceCaps
                               (
                               Printer()->Canvas->Handle,
                               LOGPIXELSX
                               );
    float fScreenVert  = (float)GetDeviceCaps
                               (
                               this->Canvas->Handle,
                               LOGPIXELSY
                               );
    float fScreenHorz  = (float)GetDeviceCaps
                              (
                              this->Canvas->Handle,
                              LOGPIXELSX
                              );
    int iHeight = (int) ((float)Image->Picture->Bitmap->Height *
                              (fPrinterVert / fScreenVert));
    int iWidth  = (int) ((float)Image->Picture->Bitmap->Width *
                              (fPrinterHorz / fScreenHorz));

    // Печатаем растр (версия VCL).
    StretchBltBitmap(Printer()->Canvas, 0, 0, iWidth, iHeight,
                     Image->Picture->Bitmap);

    // Печать завершена.
    Printer()->EndDoc();
}
```

Джозеф Хечт (Joe C. Hecht) отправил по адресу Deja.com написанную на Delphi программу, демонстрирующую его собственный вариант преобразования палитры, не зависящий от функций

Microsoft или Borland. C++Builder-компонент TPrintDIB содержится на прилагаемом к книге компакт-диске в подпапке TPrintDIB внутри папки, отведенной для этой главы. Данная C++Builder-версия Delphi-программы Джозефа Хечта создана Дэймоном Чендлером (Damon Chandler) и Дэйвом Ричардом (Dave Richard). Этот компонент обеспечивает даже более качественную обработку палитры, чем реализация компании Borland, хотя мне и не удалось обнаружить никаких различий в моих контрольных распечатках. У Джозефа Хечта также есть собственная улучшенная версия той же программы, именуемая TExcellentImagePrinter, которая относится к разряду условно-бесплатных программ. Ее можно найти по адресу http://www.code4sale.com/joehecht. Ниже приводится фрагмент проекта PrintSample4, демонстрирующий использование компонента TPrintDIB. Раздел, в котором выполняется масштабирование, опущен, поскольку он аналогичен соответствующему коду в листинге 23.10. Перед открытием проекта PrintSample4 необходимо инсталлировать компонент TPrintDIB в среде C++Builder.

```
// Печать растрового изображения (версия TPrintDIB).
PrintDIB->StretchDrawDIB(Printer()->Canvas->Handle, 0, 0,
                         iWidth, iHeight, Image->Picture->Bitmap);
```

Печать других типов графических изображений с использованием VCL

Простой способ печати нерастрового изображения — загрузить его в экземпляр класса TPicture. Это можно сделать либо с помощью компонента TImage, либо динамически, создав экземпляр класса TPicture, а затем скопировав содержимое изображения в экземпляр класса TBitmap и распечатав этот растр, используя описанную выше функцию StretchBltBitmap() или экземпляр класса TPrintDIB. В листинге 23.11 приведен пример печати изображения, загруженного в объект TImage с именем Image1:

Листинг 23.11. Печать нерастровых изображений

```
// Экземпляр класса Tbitmap, который будет содержать преобразованное изображение.
    Graphics::TBitmap *pBitmap = new Graphics::TBitmap();

    try
    {
        // Подготовка растра.
        pBitmap->Width  = Image1->Picture->Width;
        pBitmap->Height = Image1->Picture->Height;
        pBitmap->PixelFormat = pf24bit;

        // Преобразование нерастрового изображения в растровое.
        pBitmap->Canvas->Draw(0,0, Image1->Picture->Graphic);

        // Печать изображения.
        Printer()->BeginDoc();
        StretchBltBitmap(Printer()->Canvas, 0, 0, 3000, 3000,
                         pBitmap);
        Printer()->EndDoc();
    }
    catch(...) { }

    // Освобождение экземпляра.
    delete pBitmap;
```

Использование более сложных методов печати

На прилагаемом к книге компакт-диске (в папке, выделенной для этой главы) находится программа Device Info, которая может пригодиться, если вас заинтересует информация о конкретном устройстве. Эта программа использует функцию Win32 API GetDeviceCaps(), описание которой приводится в следующем разделе.

Определение разрешения принтера

Разрешение принтера (ему также соответствует термин *качество печати*) измеряется в пикселях на дюйм (pixels per inch — ppi). Существует два способа получить эту информацию.

- С помощью функции GetDeviceCaps() с параметрами LOGPIXELSX и LOGPIXELSY для получения значений горизонтального и вертикального разрешения.

```
int iLogPixelsX = GetDeviceCaps(Printer()->Handle, LOGPIXELSX);
int iLogPixelsY = GetDeviceCaps(Printer()->Handle, LOGPIXELSY);
```

- С помощью свойства PixelsPerInch класса TFont. Обратите внимание, что информация, возвращаемая свойством PixelsPerInch, достоверна только в том случае, когда значения горизонтального и вертикального разрешения совпадают (в некоторых струйных принтерах эти значения различны), поскольку свойство PixelsPerInch возвращает то же значение, что и функция GetDeviceCaps() с параметром LOGPIXELSY. Не следует забывать, что принтер должен находится в состоянии печати, а функция BeginDoc() должна быть вызвана для объекта Printer() до попытки получить доступ к свойству принтера Canvas.

```
int iPixelsPerInch = Printer()->Canvas->Font->PixelsPerInch;
```

Определение размера печатной области бумаги

Невозможно печатать на всей площади бумаги, поэтому весьма полезно знать размеры печатной и непечатной областей. Информацию о текущем размере бумаги можно получить как в пикселях, так и миллиметрах.

Определение размеров в пикселях

- Для получения вертикальных и горизонтальных размеров в пикселях используйте функцию GetDeviceCaps() с параметрами HORZRES и VERTRES.

```
int iHorzRes = GetDeviceCaps(Printer()->Handle, HORZRES);
int iVertRes = GetDeviceCaps(Printer()->Handle, VERTRES);
```

- Используйте свойства PageWidth и PageHeight класса TPrinter, которые на самом деле представляют собой инкапсуляцию двух вызовов функций, представленных в первом методе.

```
int iHorzRes = Printer()->PageWidth;
int iVertRes = Printer()->PageHeight;
// NB: В примере использования свойств PageWidth и PageHeight,
// приведенных в справочной системе Borland, есть ошибка.
```

```
// Строку
// TPrinter Prntr = Printer();
// нужно заменить строкой
// TPrinter *Prntr = Printer();
```

Определение размеров в миллиметрах

Для получения размера бумаги в миллиметрах достаточно использовать вызовы функций, представленные в первом методе, но параметры HORZRES и VERTRES нужно заменить параметрами HORZSIZE и VERTSIZE, соответственно.

Определение физического размера бумаги

Существует два вида информации, связанной с физическими размерами бумаги. Во-первых, это реальные размеры бумаги, которые почти всегда больше размеров печатной области. Во-вторых, это физическое смещение, которое представляет собой расстояния от левого и верхнего краев бумаги до начала печатной области. Результаты возвращаются в пикселях.

Физический размер бумаги

Для получения физического размера бумаги вызовите функцию GetDeviceCaps() с параметрами PHYSICALWIDTH и PHYSICALHEIGHT.

Физическое смещение

Различают следующих два вида смещения.

- Расстояние от левого края бумаги до левого края печатной области. Для получения этой информации используйте функцию GetDeviceCaps() с параметром PHYSICALOFFSETX.
- Расстояние от верхнего края бумаги до верхнего края печатной области. Для получения этой информации используйте функцию GetDeviceCaps() с параметром PHYSICALOFFSETY.

Определение возможностей принтера в области рисования

Установить, на что способен принтер в качестве устройства воспроизведения рисунков, снова поможет функция GetDeviceCaps(). Эта информация оказывается полезной, если вашему приложению нужен принтер для поддержки на должном уровне различных GDI-функций. Тот факт, что ваш принтер не полностью поддерживает все функции, совсем не означает, что вам нельзя их использовать. Интерфейс GDI эмулирует их, за исключением функций BitBlt(), StretchBlt() или DIBStretchBlt() на принтерах, которые не поддерживают пикселей растровых строк (например, на перьевых графопостроителях).

Выделяют пять основных категорий сбора информации.

- RASTERCAPS. Содержит информацию о том, какие растровые функции (например, StretchDIBits) поддерживает принтер.

```
if (GetDeviceCaps(Printer()->Handle, RASTERCAPS) & RC_STRETCHDIB)
   // Принтер поддерживает функцию StretchDIBits.
else
   // Принтер не поддерживает функцию StretchDIBits.
```

- CURVECAPS. Содержит информацию о том, какие функции начертания кривых (например, окружности) поддерживает принтер.

```
if (GetDeviceCaps(Printer()->Handle, CURVECAPS) & CC_CIRCLES)
   // Принтер поддерживает рисование окружностей (CIRCLES).
else
   // Принтер не поддерживает рисование окружностей (CIRCLES).
```

- LINECAPS. Содержит информацию о том, какие функции начертания прямых поддерживает принтер (например, полилинии, или составные линии, т.е. состоящие из ряда связанных линейных сегментов).

```
if (GetDeviceCaps(Printer()->Handle, LINECAPS) & LC_POLYLINE)
   // Принтер поддерживает рисование полилиний (PolyLines).
else
   // Принтер не поддерживает рисование полилиний (PolyLines).
```

- POLYGONALCAPS. Содержит информацию о том, какие функции начертания многоугольников (например, прямоугольники) поддерживает принтер.

```
if (GetDeviceCaps(Printer()->Handle, POLYGONALCAPS) &

                                PC_RECTANGLE)
    // Принтер поддерживает рисование прямоугольников.
else
    // Принтер не поддерживает рисование прямоугольников.
```

- TEXTCAPS. Содержит информацию о том, какие функции начертания текста (например, *курсив*) поддерживает принтер.

```
if (GetDeviceCaps(Printer()->Handle, TEXTCAPS) & TC_IA_ABLE)
    // Принтер поддерживает курсив.
else
    // Принтер не поддерживает курсив.
```

Как напечатать повернутым шрифтом

Пример, приведенный в листинге 23.12, демонстрирует возможность печати повернутым шрифтом (предполагается, что принтер поддерживает печать повернутых символов и шрифт TrueType).

Листинг 23.12. Печать повернутым шрифтом

```
// --- Полностью исходный код приведен в проекте PrintSample5 ---
void __fastcall TMainForm::PrintWithRotatedFont()
{
    // --- Локальные переменные. --- //
    TLogFont logFont;           // VCL-инкапсуляция WIN32-структуры LOGFONT.
                                // Структура LOGFONT определяет атрибуты шрифта.
    HFONT hOldFont;             // Дескриптор для шрифта (до применения
                                // атрибутов структуры LOGFONT).
    HFONT hNewFont;             // Дескриптор для шрифта (после применения
                                // атрибутов структуры LOGFONT).

    // Начало печати.
```

```
    Printer()->BeginDoc();
    Printer()->Canvas->Font->Name = "Arial";
    Printer()->Canvas->Font->Size = 24;

    // Создаем экземпляр класса TLogFont из текущего шрифта канвы принтера.
    GetObject(Printer()->Canvas->Font->Handle,
              sizeof(logFont),
              &logFont);
    // Устанавливаем поворот на 45° против часовой стрелки.
    logFont.lfEscapement = 450;
    logFont.lfOrientation = 450;

    // Применяем атрибуты структуры LOGFONT к текущему шрифту и
    // сохраняем его дескриптор в переменной hNewFont.
    hNewFont = CreateFontIndirect(&logFont);
    // Сохраняем дескриптор старого шрифта (текущего шрифта канвы принтера)
    // и устанавливаем шрифт канвы принтера равным новому шрифту.
    hOldFont = SelectObject(Printer()->Canvas->Handle, hNewFont);

    // Выводим на принтер текст, используя повернутый шрифт.
    Printer()->Canvas->TextOut(
            // 2.5" вправо и 3.0" вниз
            int(2.5*GetDeviceCaps(Printer()->Canvas->Handle,
                                  LOGPIXELSY)),
            int(3.0*GetDeviceCaps(Printer()->Canvas->Handle,
                                  LOGPIXELSY)),
            "Повернутый текст!");

    // Восстанавливаем старый шрифт в качестве текущего шрифта канвы принтера.
    SelectObject(Printer()->Canvas->Handle, hOldFont);
    // Удаляем наш экземпляр нового шрифта.
    DeleteObject(hNewFont);

    // Печать завершена.
    Printer()->EndDoc();
}
```

Получение доступа к параметрам принтера

Ключевой “фигурой” в теме доступа к параметрам принтера является Win32-структура DEVMODE, которая содержит информацию об установке устройства, данные, которые можно изменять при переходе от одной распечатки к другой, количество печатаемых копий и пр. К структуре DEVMODE можно получить доступ с помощью различных функций. Ниже будут описаны такие Win32-функции, как DocumentProperties(), GetPrinter(), SetPrinter() и DeviceCapabilities(), а также методы класса TPrinter GetPrinter() и SetPrinter(). Win32-функции GetPrinter() и SetPrinter() позволяют получить даже более подробную информацию, чем из различного рода структур PRINTER_INFO. Для получения доступа к данным, предназначенным только для чтения (имена источников бумаги, имена форматов бумаги, размеры бумаги), используйте функцию DeviceCapabilities().

Итак, рассмотрим назначение следующих функций.

- `DeviceCapabilities()`. Эта функция считывает характеристики драйвера принтера.
- `DocumentProperties()`. Эта функция считывает или модифицирует информацию по инициализации принтера или отображает таблицу параметров конфигурации для заданного принтера.
- Методы `GetPrinter()` и `SetPrinter()` класса `TPrinter`. Они представляют собой `TPrinter`-инкапсуляцию функции `DocumentProperties()`. Библиотека VCL также содержит инкапсуляцию для диалогового окна Print Setup (`TPrinterSetupDialog`), которое может отобразить функция `DocumentProperties()`. Если есть необходимость в использовании методов класса `TPrinter` во время печати, например методов класса `TCanvas`, вместо функции `DocumentProperties()` для настройки параметров печати используйте функцию `GetPrinter()`, поскольку функция `DocumentProperties()` требует, чтобы сначала (для получения дескриптора принтера) была вызвана функция `OpenPrinter()`. Нельзя использовать свойство `Handle` класса `TPrinter`, поскольку в этом случае пришлось бы иметь дело с двумя различными дескрипторами, которые не могут работать вместе. Используйте функции `GetPrinter()` и `SetPrinter()`, если вам нужно прочитать или записать содержимое структуры `DEVMODE` для принтера. Функция `SetPrinter()` применяется только для обновления свойства `Capabilities` класса `TPrinter`. Функция `SetPrinter()` не изменяет ни одного значения, устанавливаемого по умолчанию. Для изменения таких значений используйте Win32-функцию `SetPrinter()` и структуру `PRINTER_INFO_2`. Подробности ниже.
- Win32-функция `GetPrinter()`. Эта функция считывает информацию о заданном принтере, которая возвращается на различных уровнях, задаваемых в вызове переменной уровня. Windows 95/98 поддерживает уровни 1, 2 и 5, а Windows NT/2000 — уровни 1–9. Результат возвращается в структуре уровня, как, например, в структуре `PRINTER_INFO_2` для уровня 2.
- Win32-функция `SetPrinter()`. Эта функция устанавливает данные для заданного принтера или состояние заданного принтера, приостанавливая или возобновляя процесс печати либо удаляя все задания печати. Windows 95/98 поддерживает уровни 1, 2 и 5, а Windows NT/2000 — уровни 0, 2–5 и 7–9.

Методы `GetPrinter()` и `SetPrinter()` класса `TPrinter` не следует путать с Win32-функциями `GetPrinter()` и `SetPrinter()`.

Для получения подробной информации о содержимом структур `DEVMODE` и `PRINTER_INFO` следует обратиться к справочнику по интерфейсу Win32. Упомянутые функции и методы рассматриваются далее в этом разделе наряду с менее популярными API-функциями.

Как узнать имя принтера, используемого по умолчанию

В классе `TPrinter` предусмотрена инкапсуляция для получения имени стандартного принтера. Чтобы прочитать имя принтера, используемого по умолчанию, установите свойство `PrinterIndex` равным `-1` и вызовите метод `GetPrinter()` класса `TPrinter`.

```
char szDeviceName[CCHDEVICENAME],    // Имя принтера.
         szDriverName[MAX_PATH],     // Фиктивная переменная.
         szPortName[MAX_PATH];       // Фиктивная переменная.
    THandle hPrnDevMode;             // Фиктивная переменная.

    // Выбираем стандартный принтер.
    Printer()->PrinterIndex = -1;

    // Получаем имя стандартного принтера (szDeviceName
    // сейчас содержит имя стандартного принтера).
    Printer()->GetPrinter(szDeviceName, szDriverName, szPortName,
                          hPrnDevMode);
```

Имя принтера также можно прочитать прямо из свойства `Printers` (до вызова этого свойства нужно установить свойство `PrinterIndex` равным `-1`).

```
Printer()->Printers->Strings[Printer()->PrinterIndex];
```

Установка стандартного принтера

Существует три способа установки стандартного принтера, которые рассматриваются ниже и демонстрируются в листингах 23.13, 23.14 и 23.15. Полное описание исходного кода можно найти в подпапке `Samples\SetDefaultPrinter` папки, отведенной для данной главы.

Как упоминалось выше, структуры `PRINTER_INFO` используются для чтения (с помощью Win32-функции `GetPrinter()`) и записи (с помощью Win32-функции `SetPrinter()`) информации о принтере.

Первый метод (он годится для всех версий Windows) состоит в записи информации в файл `WIN.INI`. Несмотря на то что файл `WIN.INI` можно считать пережитком прошлого, т.е. "эпохи" Windows 3.x, он по-прежнему используется в некоторых случаях под управлением новых версий Windows. Одним из таких случаев и является чтение и запись информации для стандартного принтера. Строка `Profile` (из раздела `PrinterPorts` в `INI`-файле) в среде Windows 9x имеет следующий вид:

```
HP LaserJet 4P,HPPCL5MS,LPT1:
```

А в среде Windows NT/2000 немного иной:

```
HP LaserJet 4P,winspool,LPT1:
```

Нет смысла запоминать все это, поскольку можно получить доступ к нужной информации с помощью функции `GetPrinter()`. Однако по некоторым причинам функция `GetPrinter()` возвращает различные результаты в средах Windows 9x и Windows NT/2000, причем ни один из них не отличается полнотой. Часто либо имя драйвера, либо имя порта оказывается пустым. Вот почему имеет смысл привести пример (включенный в листинг 23.13), который позволит вам самим считать всю информацию из `INI`-файла до установки стандартного принтера.

Листинг 23.13. Установка стандартного принтера с помощью `INI`-файла

```
void __fastcall TMainForm::SetDefaultPrinterINI()
{
    char szDeviceName[CCHDEVICENAME], // Имя принтера.
         szDriverName[MAX_PATH],      // Имя драйвера.
         szPortName[MAX_PATH];        // Имя_порта/Файл/UNC-путь.
```

```
THandle hPrnDevMode;              // Дескриптор области памяти,
                                  // содержащей структуру DEVMODE.

// Читаем информацию о принтере (таким способом мы избежим
// необходимости удалять любые возможные строки "On LPT1:"
// (передача которых также возможна на другие системы).
// Читаем имя.
Printer()->GetPrinter(szDeviceName, szDriverName,
                      szPortName, hPrnDevMode);
if (!strlen(szPortName) || !strlen(szDriverName))
{
    // --- Если szPortName или szDriverName пустует,
    //                                  заполняем их.---
    char szTemp[MAX_PATH];
    GetProfileString("Devices", szDeviceName, "",
                     szTemp, MAX_PATH);

    // Находим индекс следующей запятой.
    char *pszPos = StrPos(szTemp, ",");
    if (pszPos)
    {
        // Читаем имя драйвера.
        int iLength = strlen(szTemp)-strlen(pszPos);
        strncpy(szDriverName, szTemp, iLength);
        // Ограничиваем строку.
        szDriverName[iLength] = '\0';
        // Читаем имя порта.
        strcpy(szPortName, ++pszPos);
    }  // ! if
}  // ! if

// Создаем строку профиля для нового стандартного принтера.
AnsiString asProfileString = AnsiString(szDeviceName) + "," +
                             szDriverName + "," + szPortName;

// Облегчая жизнь пользователю, отображаем строку профиля в заголовке.
Caption = Application->Title + " [ProfileString: \ "" +
          asProfileString + "\"]";

// Записываем строку профиля в файл WIN.INI (устанавливаем
// новый стандартный принтер).
WriteProfileString(TEXT("Windows"),TEXT("Device"),
                   asProfileString.c_str() );

// В зависимости от OS посылаем корректное сообщение.
 // BROADCAST
if (IsNT())
{
 // Новый тип сообщения для NT/W2k.
    SendMessageTimeout(HWND_BROADCAST, WM_WININICHANGE, 0L,
```

```
                        0L, SMTO_NORMAL, 1000, NULL);
    }
    else
    {
        // Старый тип сообщения для Windows 9x.
        SendMessageTimeout(HWND_BROADCAST, WM_WININICHANGE, 0L,
                        (LPARAM)(LPCTSTR)"windows", SMTO_NORMAL,
                         1000, NULL);
    }
}
```

Второй метод (он подходит только для Windows 95/98) состоит в использовании Win32-функции `SetPrinter()` и структуры `PRINTER_INFO_5`. Сначала вызываем функцию `GetPrinter()`, чтобы вычислить объем памяти, необходимый для заполнения структуры `PRINTER_INFO_5`. (Здесь можно было бы также использовать структуру `PRINTER_INFO_2`, но структура `PRINTER_INFO_5` меньше, а нам как раз и не нужна дополнительная информация, содержащаяся в структуре `PRINTER_INFO_2`.) Затем выделяем эту память и вызываем функцию `GetPrinter()` для заполнения структуры `PRINTER_INFO_5`. После этого устанавливаем атрибут для стандартного принтера и фиксируем это изменение путем вызова функции `SetPrinter()`. Описанный алгоритм отражен в листинге 23.14.

Листинг 23.14. Установка стандартного принтера с помощью функции `SetPrinter()` и структуры `PRINTER_INFO_5`

```
bool __fastcall TMainForm::SetDefaultPrinter9x
                                  (char *pszPrinterName)
{
    HANDLE hPrinter;            // Дескриптор принтера.
    BOOL   bSucces = TRUE;      // Флаг успешного завершения.

    // --- Открываем принтер и читаем для него дескриптор. ---
    if(OpenPrinter(pszPrinterName, &hPrinter, NULL))
    {
        // Информация о принтере (примечание: структуру PRINTER_INFO_2 также
        // можно использовать, но структура PRINTER_INFO_5 просто меньше).
        LPPRINTER_INFO_5 printerInfo;
        DWORD dwBytesNeeded = 0;

        // Читаем необходимые байты.
        GetPrinter(hPrinter, 5, NULL, 0, &dwBytesNeeded);
        if (dwBytesNeeded &&
            (printerInfo = (LPPRINTER_INFO_5)
                                malloc(dwBytesNeeded)) != NULL)
        {
            // --- Заполняем структуру содержимым старых
            // параметров принтера. ---
            if (GetPrinter(hPrinter, 5, (LPBYTE) printerInfo,
                           dwBytesNeeded, &dwBytesNeeded))
            {
                // Устанавливаем атрибут по умолчанию.
                printerInfo->Attributes |=
                                    PRINTER_ATTRIBUTE_DEFAULT;
```

```
                // Обновляем параметры принтера.
                if (SetPrinter(hPrinter, 5,
                               (LPBYTE) printerInfo, 0))
                {
                    bSucces = TRUE;
                }
                else
                {
                    bSucces = FALSE;
                }  // ! if SetPrinter()
            }
            else
            {
                bSucces = FALSE;
            }  // ! if GetPrinter()

            // Освобождаем память.
            free(printerInfo);
        }
        else
        {
            bSucces = FALSE;
        }  // ! if malloc()

        // Закрываем принтер.
        ClosePrinter(hPrinter);
        return bSucces;
    }
    else
    {
        return FALSE;
    }  // ! if OpenPrinter()
}
```

В третьем методе используется Win32-функция `SetDefaultPrinter()`, которая доступна только в Windows 2000.

```
bool __fastcall TMainForm::SetDefaultPrinter2K(

   char *pszPrinterName)
{
    // Только в Windows 2000 (требует инсталляции BCB5 или
    // Windows 2000 SDK).
    return SetDefaultPrinter (pszPrinterName)
}
```

Восстановление объекта `TPrinter`

Иногда объект `Printer()` нуждается во внутренней загрузке структуры `DEVMODE` для выбранного принтера. Это можно сделать, вызвав функцию `SetPrinter()` с параметром `Thandle`, равным нулю.

```
void ResetTPrinter()
{
    char szDeviceName[CCHDEVICENAME],     // Фиктивная переменная
                                          // (имя устройства).
         szDriverName[MAX_PATH],          // Фиктивная переменная
                                          // (имя драйвера).
         szPortName[MAX_PATH];            // Фиктивная переменная (имя порта).
    THandle hPrnDevMode;                  // Фиктивная переменная (DEVMODE).

    Printer()->GetPrinter(szDeviceName, szDriverName, szPortName,
                                    hPrnDevMode);
    Printer()->SetPrinter(szDeviceName, szDriverName,
                                    szPortName, 0);
}
```

Информация о доступе к структуре DEVMODE с помощью класса TPrinter

Прежде чем перейти к более сложным темам, рассмотрим реализацию доступа к структуре DEVMODE (или TDevMode, как именуется ее VCL-инкапсуляция) для принтера. Чтобы получить указатель на структуру DEVMODE для принтера, необходимо сначала вызвать метод GetPrinter() класса TPrinter для получения указателя на объект памяти, который содержит эту структуру. Затем вызываем Win32-функцию GlobalLock() для получения указателя на первый байт объекта памяти, который и представляет собой структуру DEVMODE, как показано в листинге 23.15.

Листинг 23.15. Пример использования структуры DEVMODE и функции GetPrinter() класса TPrinter

```
char szDeviceName[CCHDEVICENAME],     // Имя принтера.
         szDriverName[MAX_PATH],      // Фиктивная переменная.
         szPortName[MAX_PATH];        // Фиктивная переменная.
    THandle hPrnDevMode;              // Фиктивная переменная.
    PDEVMODE pDevMode;                // Указатель на структуру DeviceMode.

    // Получаем имя стандартного принтера (сейчас свойство
    // szDeviceName содержит имя стандартного принтера).
    Printer()->GetPrinter(szDeviceName, szDriverName,
                          szPortName, hPrnDevMode);

    if (hPrnDevMode!=0)
    {
        // Получаем указатель на структуру DEVMODE.
        pDevMode =  (PDEVMODE) GlobalLock((HANDLE)hPrnDevMode);
        if (pDevMode == NULL)
            throw Exception("Не удалось получить указатель "
                            "на структуру DEVMODE");

        // Здесь содержится код для работы со структурой DEVMODE.
```

```
        //...
        //...
        // Пример установки параметров принтера.
        pDevMode->dmFields |= DMBIN_MANUAL;
        pDevMode->dmDefaultSource = DMBIN_MANUAL;

        // Наконец, необходимо освободить структуру DEVMODE.
        GlobalUnlock((HANDLE)hPrnDevMode);

        // Печать текста с новыми параметрами принтера.
        Printer()->BeginDoc();
        Printer()->Canvas->TextOut(0,0,"Привет, мир!");
        Printer()->EndDoc();
    }
    else
        throw Exception("Не удалось получить дескриптор принтера!");
```

Эта программа изменяет параметры только для текущего принтера и только до тех пор, пока не будет изменено свойство `PrinterIndex`. Она не изменяет системные стандартные параметры, даже если вызвать функцию `SetPrinter()` класса `TPrinter` после изменения структуры `DEVMODE`. Функция `SetPrinter()` используется только для обновления свойства `Capability` класса `TPrinter`. Для изменения стандартных параметров необходимо использовать указатель на структуру `DEVMODE`. К нему можно получить доступ, используя структуру `PRINTER_INFO_2` для принтера, для которого вы собираетесь установить параметры.

Использование структуры PRINTER_INFO_2

Помимо структуры `DEVMODE`, ценная информация о принтере содержится и в структуре `PRINTER_INFO_2`. Здесь можно найти сведения об имени сервера, к которому подключен принтер (если это имеет место), действительное имя принтера (не всегда совпадающее со значением `pDeviceName` в структуре `DEVMODE`), количество заданий печати, находящихся в очереди, состояние самого принтера и пр. Ниже приводится полное содержимое структуры `PRINTER_INFO_2`.

```
typedef struct _PRINTER_INFO_2 {
  LPTSTR    pServerName;
  LPTSTR    pPrinterName;
  LPTSTR    pShareName;
  LPTSTR    pPortName;
  LPTSTR    pDriverName;
  LPTSTR    pComment;
  LPTSTR    pLocation;
  LPDEVMODE pDevMode;
  LPTSTR    pSepFile;
  LPTSTR    pPrintProcessor;
  LPTSTR    pDatatype;
  LPTSTR    pParameters;
  PSECURITY_DESCRIPTOR pSecurityDescriptor;
  DWORD     Attributes;
  DWORD     Priority;
```

```
  DWORD     DefaultPriority;
  DWORD     StartTime;
  DWORD     UntilTime;
  DWORD     Status;
  DWORD     cJobs;
  DWORD     AveragePPM;
}  PRINTER_INFO_2, *PPRINTER_INFO_2;
```

Как видите, эта структура содержит очень полезную информацию о принтере. В листинге 23.16 содержится пример чтения и записи комментариев для принтера. Работа с этой структурой не отличается от работы с другими структурами PRINTER_INFO.

Листинг 23.16. Пример чтения и записи комментариев для принтера

```
AnsiString __fastcall TMainForm::GetComment(char *szPrinterName)
{
    HANDLE              hPrinter;   // Дескриптор принтера. (Используется
    PRINTER_DEFAULTS    pd;         // только в среде Windows NT/2000
                                    // и игнорируется в среде Windows 9x).
    DWORD               dwNeeded;   // Размер структуры
                                    // PRINTER_INFO_2 в байтах.
    PRINTER_INFO_2      *pPrtInfo2; // Указатель на структуру PRINTER_INFO_2.
    AnsiString          asComment;  // Комментарии, собранные из
                                    // структуры PRINTER_INFO_2.

    // Инициализируем структуры принтера стандартными значениями.
    ZeroMemory(&pd, sizeof(PRINTER_DEFAULTS));
    // Получаем полный доступ.
    pd.DesiredAccess = PRINTER_ALL_ACCESS;

    // Открываем принтер.
    if(!OpenPrinter(szPrinterName, &hPrinter, &pd))
    {
       // Функция OpenPrinter() не достигла своей цели.
       throw Exception("Вызов функции OpenPrinter() неуспешен.");
    }

    // Получаем размер структуры PRINTER_INFO_2
    // (2 означает уровень).
    if(!GetPrinter(hPrinter, 2, NULL, 0, &dwNeeded))
    {
        // Если ошибка не равна значению
        // ERROR_INSUFFICIENT_BUFFER, нужно ретироваться.
        if(GetLastError() != ERROR_INSUFFICIENT_BUFFER)
        {
           ClosePrinter(hPrinter);
           throw Exception(
                  "1-й вызов функции GetPrinter() потерпел неудачу.");
        }
    }

    // Выделяем достаточный объем памяти для структуры
    // PRINTER_INFO_2.
```

```
    pPrtInfo2 = (PRINTER_INFO_2*) malloc(dwNeeded);
    if(pPrtInfo2 == NULL)
    {
       // Провал функции malloc().
       ClosePrinter(hPrinter);
       throw Exception("Вызов функции malloc() неудачен.");
    }

    // Заполняем структуру PRINTER_INFO_2 информацией.
    if(!GetPrinter(hPrinter, 2, (LPBYTE)pPrtInfo2,
                   dwNeeded, &dwNeeded))
    {
       // Второй вызов функции GetPrinter() неудачен.
       free(pPrtInfo2);
       ClosePrinter(hPrinter);
       throw Exception("2-й вызов функции GetPrinter() неудачен.");
    }

    // Читаем комментарии.
    asComment = pPrtInfo2->pComment;

    // Освобождаем память.
    free(pPrtInfo2);
    ClosePrinter(hPrinter);

    // Возвращаем комментарии.
    return asComment;
}
//---------------------------------------------------------------
void __fastcall TMainForm::SetComment(char *szPrinterName,
                                      AnsiString &asComment)
{
    HANDLE              hPrinter;   // Дескриптор принтера.
    PRINTER_DEFAULTS    pd;         // (Используется только в среде
                                    // Windows NT/2000 и игнорируется
                                    // в среде Windows 9x).
    DWORD               dwNeeded;   // Размер структуры
                                    // PRINTER_INFO_2 в байтах.
    PRINTER_INFO_2      *pPrtInfo2; // Указатель на структуру PRINTER_INFO_2.

// Инициализируем структуры принтера стандартными значениями.
    ZeroMemory(&pd, sizeof(PRINTER_DEFAULTS));
    // Получаем полный доступ.
    pd.DesiredAccess = PRINTER_ALL_ACCESS;

    // Открываем принтер.
    if(!OpenPrinter(szPrinterName, &hPrinter, &pd))
    {
       // Провал функции OpenPrinter().
       throw Exception("Вызов функции OpenPrinter() неудачен.");
    }
```

```
    // Получаем размер структуры PRINTER_INFO_
    // (2 означает уровень).
    if(!GetPrinter(hPrinter, 2, NULL, 0, &dwNeeded))
    {
        // Если ошибка не равна значению
        // ERROR_INSUFFICIENT_BUFFER, нужно ретироваться.
        if(GetLastError() != ERROR_INSUFFICIENT_BUFFER)
        {
           ClosePrinter(hPrinter);
           throw Exception("1-й вызов функции GetPrinter() неудачен.");
        }
    }

    // Выделяем достаточный объем памяти для структуры PRINTER_INFO_2.
    pPrtInfo2 = (PRINTER_INFO_2*) malloc(dwNeeded);
    if(pPrtInfo2 == NULL)
    {
       // Провал функции malloc().
       ClosePrinter(hPrinter);
       throw Exception("Вызов функции malloc() неудачен.");
    }

    // Заполняем структуру PRINTER_INFO_2 информацией.
    if(!GetPrinter(hPrinter, 2, (LPBYTE)pPrtInfo2,
                   dwNeeded, &dwNeeded))
    {
       // Второй вызов функции GetPrinter() неудачен.
       free(pPrtInfo2);
       ClosePrinter(hPrinter);
       throw Exception(
                              "2-й вызов функции GetPrinter() неудачен.");
    }

    // Записываем комментарии.
    pPrtInfo2->pComment = asComment.c_str();

    // Обновляем объект Printer новой информацией.
    if(!SetPrinter(hPrinter, 2, (LPBYTE)pPrtInfo2, 0))
    {
       // Вызов функции SetPrinter() неудачен.
       free(pPrtInfo2);
       ClosePrinter(hPrinter);
       throw Exception("Вызов функции SetPrinter() неудачен.");
    }

    // Освобождаем память.
    free(pPrtInfo2);
    ClosePrinter(hPrinter);
}
```

Как упоминалось выше, с помощью метода GetPrinter() класса TPrinter невозможно установить системные значения параметров, которые действуют по умолчанию. Для этого необходимо использовать структуру PRINTER_INFO_2, как показано в листинге 23.17.

Листинг 23.17. Настройка принтера на работу с устройством ручной подачи бумаги

```
void SetDefaultPaperSourceToManualFeed(char *szPrinterName)
{
    HANDLE              hPrinter;     // Дескриптор принтера.
    PRINTER_DEFAULTS    pd;           // (Используется только в
                                      // среде Windows NT/2000
                                      // и игнорируется в
                                      // среде Windows 9x).
    DWORD               dwNeeded;     // Размер структуры
                                      // PRINTER_INFO_2 в байтах.
    PRINTER_INFO_2      *pPrtInfo2;   // Указатель на структуру
                                      // PRINTER_INFO_2.

    // Инициализируем структуры принтера стандартными значениями.
    ZeroMemory(&pd, sizeof(PRINTER_DEFAULTS));
    // Получаем полный доступ.
    pd.DesiredAccess = PRINTER_ALL_ACCESS;

    // Открываем принтер.
    if(!OpenPrinter(szPrinterName, &hPrinter, &pd))
    {
       // Функция OpenPrinter() потерпела неудачу.
       throw Exception("Вызов функции OpenPrinter() неудачен");
    }

    // Получаем размер структуры PRINTER_INFO_2
    // (2 означает уровень).
    if(!GetPrinter(hPrinter, 2, NULL, 0, &dwNeeded))
    {
        // Если ошибка не равна значению
        // ERROR_INSUFFICIENT_BUFFER, нужно ретироваться.
        if(GetLastError() != ERROR_INSUFFICIENT_BUFFER)
        {
           ClosePrinter(hPrinter);
           throw Exception("1-й вызов функции GetPrinter() неудачен.");
        }
    }

    // Выделяем достаточный объем памяти для структуры PRINTER_INFO_2.
    pPrtInfo2 = (PRINTER_INFO_2*) malloc(dwNeeded);
    if(pPrtInfo2 == NULL)
    {
       // Функция malloc() завершилась неудачей.
       ClosePrinter(hPrinter);
       throw Exception("Вызов функции malloc() неудачен.");
    }
```

```
    // Заполняем структуру PRINTER_INFO_2 информацией.
    if(!GetPrinter(hPrinter, 2, (LPBYTE)pPrtInfo2,
                  dwNeeded, &dwNeeded))
    {
       // Второй вызов функции GetPrinter() неудачен.
       free(pPrtInfo2);
       ClosePrinter(hPrinter);
       throw Exception("2-й вызов функции GetPrinter() неудачен.");
    }

    // Устанавливаем источник бумаги равным значению DMBIN_MANUAL
    pPrtInfo2->pDevMode->dmFields |= DMBIN_MANUAL;
    pPrtInfo2->pDevMode->dmDefaultSource = DMBIN_MANUAL;

    // Обновляем объект Printer с новой информацией.
    if(!SetPrinter(hPrinter, 2, (LPBYTE)pPrtInfo2, 0))
    {
       // Функция SetPrinter() завершилась неудачно.
       free(pPrtInfo2);
       ClosePrinter(hPrinter);
       throw Exception("Вызов функции SetPrinter() неудачен.");
    }

    // Освобождаем память.
    free(pPrtInfo2);
    ClosePrinter (hPrinter);
}
```

Другие функции, связанные с бумагой для принтера

Метод чтения и записи структуры DEVMODE одинаков для различных типов, перечисленных в следующем разделе, поэтому я ограничился только одним примером. Доступ к структуре DEVMODE описан в предыдущих разделах.

Установка формата бумаги

В Windows предусмотрен довольно длинный список заранее определенных форматов бумаги, которые можно использовать для вывода на печать при выполнении приложений. Форматы, приведенные в табл. 23.2, являются только частью “системного” списка. Полный же перечень форматов бумаги можно найти в справочной документации по Win32, содержащей описание структуры DEVMODE.

Таблица 23.2. Форматы бумаги, предусмотренные в Windows

Значение	Описание
DMPAPER_LETTER	Letter, 8 1/2×11 дюймов
DMPAPER_LEGAL	Legal, 8 1/2×14 дюймов
DMPAPER_10X14	10×14 дюймов
DMPAPER_11X17	11×17 дюймов

Окончание табл. 23.2

Значение	Описание
DMPAPER_12X11	Windows 98, Windows NT 4.0 и более поздние: 12×11 дюймов
DMPAPER_A3	A3, 297×420 миллиметров
DMPAPER_A3_ROTATED	Windows 98, Windows NT 4.0 и более поздние: A3 повернутый, 420–297 миллиметров
DMPAPER_A4	A4, 210–297 миллиметров
DMPAPER_A4_ROTATED	Windows 98, Windows NT 4.0 и более поздние: A4 повернутый, 297–210 миллиметров
DMPAPER_A4SMALL	A4 малый, 210–297 миллиметров
DMPAPER_A5	A5, 148–210 миллиметров

Размер бумаги для принтера можно установить, используя предопределенные (встроенные) форматы (перечисленные в табл. 23.2) и следующий код.

```
// Устанавливаем размер бумаги в соответствии с форматом A4.
pDevMode->dmFields |= DM_PAPERSIZE;
pDevMode->dmPaperSize = DMPAPER_A4;
```

Устанавливаем битовый флаг поля dmField, чтобы уведомить принтер о том, какие поля подлежат изменению, и изменяем содержимое указанного поля в соответствии с новым значением. Этот фрагмент кода можно было бы вставить в листинг 23.17 для демонстрации замены параметров настройки принтера.

На заметку Член dmPaperSize можно установить равным нулю, если размер бумаги задан с помощью членов dmPaperWidth и dmPaperHeight.

Чтобы установить для принтера пользовательский (нестандартный) размер бумаги, используйте следующий код.

```
// Устанавливаем размер бумаги в соответствии с пользовательскими
// значениями длины и ширины.
pDevMode->dmFields |= DM_PAPERLENGTH;
pDevMode->dmPaperLength= 1000;      // в десятых долях миллиметра
pDevMode->dmFields |= DM_PAPERWIDTH;
pDevMode->dmPaperWidth= 1000;       // в десятых долях миллиметра
```

На заметку Такой способ установки принтера работает только в среде Windows 9x. Под управлением Windows NT/2000 необходимо создать пользовательскую форму структуры типа FORM_INFO_1 и добавить ее в объект принтера, используя функцию AddForm(). Затем нужно выбрать эту форму, используя структуру PRINTER_INFO_2. Подробнее о структуре FORM_INFO можно узнать в справочнике по Win32.

```
strcpy(pPrtInfo2->pDevMode->dmFormName, "My Form Name");
pPrtInfo2->pDevMode->dmFields |= DM_FORMNAME;
pPrtInfo2->pDevMode->dmFields &= !(
            DM_PAPERSIZE | DM_PAPERLENGTH | DM_PAPERWIDTH);
```

Получение списка поддерживаемых форматов и источников бумаги (способов подачи)

В папке Samples прилагаемого к книге компакт-диска находится проект PaperAndBin, который демонстрирует возможность получения доступа к информации о поддерживаемых форматах, а также типах и именах источников бумаги (способов подачи) для принтера. В листинге 23.18 приведена часть исходного кода из этого проекта.

Листинг 23.18. Заполнение трех строковых переменных информацией о принтере (имя, порт и драйвер)

```
void __fastcall TMainForm::FillPrinterNames()
{
    // Переменные szDeviceName, szDriverName и szPortName объявлены
    // закрытыми в классе формы (в заголовочном файле формы).
    THandle hPrnDevMode;    // Дескриптор для области памяти,
                            // содержащей структуру DEVMODE.

     // Читаем информацию о принтере (так мы избежим
    // необходимости удалять любые возможные строки "On LPT1:"
    // (передача которых также возможна на другие системы).

    // Обновляем выбранный принтер (из предложенного списка).
    Printer()->PrinterIndex = PrinterComboBox->ItemIndex;

    // Читаем имя.
    Printer()->GetPrinter(szDeviceName, szDriverName,
                          szPortName, hPrnDevMode);
    // Проверяем, не пустое ли имя порта и имя драйвера
    if (!strlen(szPortName) || !strlen(szDriverName))
    {
        // --- Одна из строковых переменных szPortName или
        // szDriverName пуста, заполняем ее. ---
        char szTemp[MAX_PATH];
GetProfileString("Devices", szDeviceName, "",
                         szTemp, MAX_PATH);
        // Находим индекс следующей запятой.
        char *pszPos = StrPos(szTemp, ",");
        if (pszPos)
        {
            int iLength = strlen(szTemp) - strlen(pszPos);
            // Читаем имя драйвера.
            strncpy(szDriverName, szTemp, iLength);
            szDriverName[iLength] = '\0'; // Ограничиваем строку.
            // Читаем имя порта.
            strcpy(szPortName,  ++pszPos);
        }  // ! if
    }  // !  if
}
```

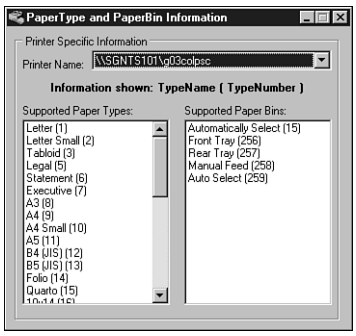


Рис. 23.3. Проект PrinterAndBin в действии

Этот проект содержит функцию FillPrinterNames(), которая представляет собой инкапсуляцию обращения к методу GetPrinter() класса TPrinter. Она корректно заполняет строковые переменные szDriverName и szPortName. Эта функция вызывается до вызова функций FillPaperInfo() и FillBinInfo(), что обеспечивает доступность необходимой информации. Каждый раз при выборе нового принтера в комбинированном списке PrinterComboBox на экране очищаются списки TListBox, и для их очередного заполнения вызываются функции FillPrinterNames(), FillPaperInfo() и FillBinInfo(). На рис. 23.3 показан интерфейс пользователя для проекта PrinterAndBin.

Получение числовых типов и имен форматов бумаги

Функция FillPaperInfo() сначала собирает информацию о числовых типах и именах форматов бумаги, а затем заполняет этой информацией список типа TListBox, именуемый PaperListBox, как показано в листинге 23.19.

Листинг 23.19. Получение числовых типов и имен форматов бумаги с последующей вставкой их в список PaperListBox

```
void __fastcall TMainForm::FillPaperInfo()
{
    // Примечание: в заголовочном файле проекта я создал const-имена
    // CCHPAPERNAME со значением 64. Об этом должны были позаботиться
    // сотрудники Microsoft, но поскольку они этого не сделали,
    // нам пришлось самим восполнить этот пробел.)

    // Примечание: переменные szDeviceName, szDriverName и szPortName
    // объявлены закрытыми (private) в классе формы (а именно в заголовочном
    // файле формы) и уже заполнены нужной для нас информацией.

    // Получаем количество форматов бумаги.
    int iNumOfPapers = DeviceCapabilities(szDeviceName,
                                          szPortName,
                                          DC_PAPERS,
                                          NULL,
                                          NULL);

    // Выделяем память для хранения списка форматов бумаги (в виде чисел).
    WORD *pwPaperTypes = new WORD[iNumOfPapers];
    // Получаем сам список форматов бумаги (в виде чисел).
    DeviceCapabilities(szDeviceName,
                       szPortName,
                       DC_PAPERS,
                       (LPTSTR)pwPaperTypes,
                       NULL);

    // Выделяем память для хранения всех имен форматов бумаги.
```

```
    char *pszPaperNames = new char[iNumOfPapers*CCHPAPERNAME];

    // Получаем сам список имен форматов бумаги.
    DeviceCapabilities(szDeviceName,
                       szPortName,
                       DC_PAPERNAMES,
                       pszPaperNames,
                       NULL);


    // Заполняем список ListBox полученной информацией.
    char *pszName;
    for (int iLoop=0; iLoop < iNumOfPapers; iLoop++)
    {
        pszName = &pszPaperNames[iLoop*CCHPAPERNAME];
        PaperListBox->Items->Add(AnsiString(pszName) +
                        " ("+IntToStr(pwPaperTypes[iLoop])+")");
    }
}
```

Получение числовых типов и имен источников бумаги

Функция `FillBinInfo()` сначала собирает информацию о числовых типах и именах источников бумаги, а затем заполняет этой информацией список типа `TListBox`, именуемый `BinListBox`, как показано в листинге 23.20.

Листинг 23.20. Получение числовых типов и имен источников бумаги с последующей вставкой их в список `BinListBox`

```
void __fastcall TMainForm::FillBinInfo()
{
    // Примечание: в заголовочном файле проекта я создал const-имена
    // CCHBINNAME со значением 24. Об этом должны были позаботиться
    // сотрудники Microsoft, но поскольку они этого не сделали,
    // нам пришлось самим восполнить этот пробел.)

    // Примечание: переменные szDeviceName, szDriverName и szPortName
    // объявлены закрытыми (private) в классе формы (а именно в заголовочном
    // файле формы) и уже заполнены нужной для нас информацией.

    // Получаем количество источников бумаги.
    int iNumOfBins = DeviceCapabilities(szDeviceName,
                                        szPortName,
                                        DC_BINS,
                                        NULL,
                                        NULL);

    // Выделяем память для хранения списка типов источников
    // бумаги (в виде чисел).
    WORD *pwBinTypes = new WORD[iNumOfBins];
    // Получаем список типов источников бумаги (в виде чисел).
```

```
    DeviceCapabilities(szDeviceName,
                       szPortName,
                       DC_BINS,
                       (LPTSTR)pwBinTypes,
                       NULL);

    // Выделяем память для хранения всех имен источников бумаги.
    char *pszBinNames = new char[iNumOfBins*CCHBINNAME];

    // Получаем список имен источников бумаги.
    DeviceCapabilities(szDeviceName,
                       szPortName,
                       DC_BINNAMES,
                       pszBinNames,
                       NULL);


    // Заполняем список ListBox полученной информацией.
    char *pszName;
    for (int iLoop=0; iLoop < iNumOfBins; iLoop++)
    {
        pszName = &pszBinNames[iLoop*CCHBINNAME];
        BinListBox->Items->Add(AnsiString(pszName)
                  +" ("+IntToStr(pwBinTypes[iLoop])+")");
    }
}
```

Установка источника бумаги для принтера

Иногда возникает необходимость в выборе источника бумаги (способа ее подачи), поскольку многие компании используют различные типы и размеры бумаги и для каждого из них предусмотрен в принтере свой способ подачи. Для того чтобы приложение могло поддерживать тот или иной тип печати, необходимо "заставить" его "распознавать" более широкий спектр источников бумаги, чем предусмотрено драйвером данного принтера. Ниже приведен список распространенных значений источников бумаги, который можно для этого использовать. В случае использования источника, определяемого конкретным драйвером, соответствующее ему значение должно быть больше или равно значению DMBIN_USER.

- DMBIN_ONLYONE
- DMBIN_LOWER
- DMBIN_MIDDLE
- DMBIN_MANUAL
- DMBIN_ENVELOPE
- DMBIN_ENVMANUAL
- DMBIN_AUTO
- DMBIN_TRACTOR
- DMBIN_SMALLFMT
- DMBIN_LARGEFMT
- DMBIN_LARGECAPACITY
- DMBIN_CASSETTE
- DMBIN_FORMSOURCE

Например, чтобы установить источник бумаги в режим ручной подачи листов, необходимо установить поля структуры DEVMODE следующим образом.

```
pPrtInfo2->pDevMode->dmFields |= DMBIN_MANUAL;
pPrtInfo2->pDevMode->dmDefaultSource = DMBIN_MANUAL;
```

Установка количества копий

Для задания количества печатаемых копий также можно использовать структуру DEVMODE. С помощью следующих строк программы количество копий устанавливается равным 10.

```
pDevMode->dmFields |= DM_COPIES;
pDevMode->dmCopies = 10;
```

Установка ориентации

Для задания ориентации печати можно использовать свойство Orientation класса TPrinter.

```
pDevMode->dmFields |= DM_ORIENTATION;
pDevMode->dmOrientation = DMORIENT_LANDSCAPE;    // Другое возможное значение:
                                                 // DMORIENT_PORTRAIT
```

Установка масштаба

Поле dmScale задает коэффициент (в процентах) масштабирования результата вывода на печать.

```
pDevMode->dmFields |= DM_SCALE;
pDevMode->dmScale = 75;
```

Установка цветового режима

Используйте поле режима dmColor для переключения цветного принтера из черно-белого режима печати в цветной и наоборот.

```
pDevMode->dmFields |= DM_COLOR;
pDevMode->dmColor = DMCOLOR_COLOR;         // Другое возможное значение:
                                           // DMCOLOR_MONOCHROME
```

Установка качества печати

Поле dmPrintQuality в структуре DEVMODE задает разрешение принтера. В табл. 23.3 приведены четыре встроенных аппаратно-независимых значения:

Таблица 23.3. Встроенные значения качества печати

Значение	Описание
DMRES_HIGH	Печать с высоким разрешением
DMRES_MEDIUM	Печать со средним разрешением
DMRES_LOW	Печать с низким разрешением
DMRES_DRAFT	Печать чернового качества

Положительное значение задает количество точек на дюйм (*d*ots *p*er *i*nch —DPI) и следовательно является аппаратно-зависимым. Чтобы указать разрешение вручную, необходимо установить поле dmYResolution в структуре DEVMODE равным разрешению по координате Y, а поле dmPrintQuality равным разрешению по координате X (в DPI).

Ниже приведен пример установки чернового качества.

```
pDevMode->dmFields |= DM_PRINTQUALITY;
pDevMode->dmPrintQuality = DMRES_DRAFT;
```

Принтер должен поддерживать установленное вами конкретное разрешение. Ниже в этом разделе мы рассмотрим, как получить список значений поддерживаемых разрешений. При выполнении следующих строк программы для принтера устанавливается разрешение печати 300×300.

```
pDevMode->dmFields |= DM_PRINTQUALITY | DM_YRESOLUTION;
pDevMode->dmPrintQuality = 300;
pDevMode->dmYResolution = 300;
```

Установка дуплексного режима

Поле dmDuplex "отвечает" за переключение в режим дуплексной (двухсторонней) печати для принтеров, способных работать в этом режиме. В табл. 23.4 описаны возможные значения этого параметра.

Таблица 23.4. Типы дуплексного режима

Значение	Описание
DMDUP_SIMPLEX	Нормальная (недуплексная) печать
DMDUP_HORIZONTAL	Связывание по короткому краю. Длинный край страницы располагается по горизонтали
DMDUP_VERTICAL	Связывание по длинному краю. Длинный край страницы располагается по вертикали

Следующие строки программы показывают, как настроить принтер на использование дуплекса по длинному краю бумаги.

```
pDevMode->dmFields |= DM_DUPLEX;
pDevMode->dmDuplex = DMDUP_VERTICAL;
```

Установка параметра Разобрать по копиям

Установка режима сортировки по копиям (Collate Mode) возможна при печати нескольких копий документа. Поле dmCollate игнорируется, если драйвер принтера не предусматривает поддержку сортировки путем установки члена dmFields равным значению DM_COLLATE. Значение DMCOLLATE_TRUE обеспечивает более быстрый и эффективный результат разбора по копиям, поскольку данные посылаются драйверу устройства только один раз, независимо от количества требуемых копий. Принтер просто выполняет команду повторно напечатать данную страницу.

```
pDevMode->dmFields |= DM_COLLATE;
pDevMode->dmCollate = DMCOLLATE_TRUE;      // Другое возможное
                                           // значение: DMCOLLATE_FALSE
```

Получение информации о возможных разрешениях принтера

Чтобы получить список возможных значений разрешения принтера, вызовите функцию DeviceCapabilities() с параметром DC_ENUMRESOLUTIONS. Сбор этой информации аналогичен описанным выше методам сбора данных об именах форматов и источников бумаги.

Получение информации о состоянии принтера

Чтобы прочитать состояние принтера, можно использовать поле `Status` структуры `PRINTER_INFO_2`. В список состояний принтера входят: Paper Jam (затор бумаги), Paper Out (отсутствие бумаги), Busy (занят) и пр. В листинге 23.13 было показано, как получить эту информацию из структуры `PRINTER_INFO_2`. Подробный пример чтения состояния принтера и задания печати можно найти в разделе "HOWTO: Get the Status of a Printer and a Print Job" (ID = Q160129), который доступен на Web-узле MSDN (сокр. от *Microsoft Developer Network* — собрание документов компании Microsoft, содержащее сведения обо всех ее разработках) по адресу: `http://msdn.microsoft.com`.

Это лишь малая часть информации, которую можно прочитать и записать, используя структуры `DEVMODE`, `PRINTER_INFO_2` и функцию `DeviceCapabilities`. Если вас интересуют моменты, не описанные здесь, загляните в разделы справочника по Win32, посвященные структурам `DEVMODE`, `PRINTER_INFO` и функции `DeviceCapabilities`. Методы чтения и записи этих данных рассматривались выше, поэтому вам достаточно распространить их на решение новых проблем.

Работа с заданиями печати

Полное обсуждение темы заданий печати выходит за рамки данной главы, поскольку перед нами поставлена цель лишь ввести вас в курс обозначенной темы. В процессе нашего "краткого курса" мы укажем источники более подробной информации, которые доступны в Internet.

Для получения списка заданий печати для данного принтера можно задействовать две различные функции: Win32-функцию `GetPrinter()`, работающую со структурой `PRINTER_INFO_2`, или функцию `EnumJobs()` (использующую структуру `JOB_INFO_1` либо `JOB_INFO_2`). Структура `PRINTER_INFO_2` содержит лишь количество заданий, находящихся в очереди к принтеру, поэтому чаще используется другой вариант, т.е. функция `EnumJobs()`. Структура `JOB_INFO_2` предоставляет больше данных, чем структура `JOB_INFO_1`, но каждая из них более информативна по сравнению со структурой `PRINTER_INFO_2`. Используя функцию `EnumJobs()` и какую-либо структуру `JOB_INFO`, можно получить информацию об ID задания, имени документа, состоянии задания, позиции в очереди, общем количестве страниц в задании, количестве уже напечатанных страниц и многое другое. О том, как правильно вызвать функцию `EnumJobs()`, читайте раздел "HOWTO: How to Call Win32 Spooler Enumeration APIs Properly" (ID = Q158828), который можно найти на Web-узле MSDN (`http://msdn.microsoft.com/support/kb/articles/Q158/8/28.asp`).

В интерфейсе Win32 API также предусмотрены функции для работы с заданиями печати: `AddJob()`, `GetJob()`, `SetJob()` и `ScheduleJob()`. Для уведомления об изменениях в очереди к принтеру Win32 предоставляет две возможности. Windows-сообщение `WM_SPOOLERSTATUS` должно работать в средах Windows 9x/NT/2000, но в действительности оказалось, что оно успешно работает только под управлением Windows 9x. Такие функции, как `FindFirstPrinterChangeNotification()`, `FindNextPrinterChangeNotification()` и `FindClosePrinterChangeNotification()` работают только в средах Windows NT/2000. Чтобы на примере понять, как создается монитор очереди к принтеру (с помощью этих API-функций и функции `EnumJobs()`), читайте в разделе "SAMPLE: PrintMon.exe Demonstrates the Win32 Spooler API" (ID = Q196805) на Web-узле MSDN.

Пример перехвата сообщения `WM_SPOOLERSTATUS` в среде C++Builder приведен в листинге 23.21. Чтобы не усложнять его, мы создаем карту сообщений, которая реализует механизм перехвата сообщения `WM_ SPOOLERSTATUS` и вызова соответствующего метода класса, используя VCL-оболочку для сообщения `WM_SPOOLERSTATUS`, именуемую `TWMSpoolerStatus`.

Добавляем код, приведенный в листинге 23.21, в заголовочный файл формы (здесь она называется TMainForm):

Листинг 23.21. Перехват сообщения WM_SPOOLERSTATUS

```
class TMainForm : public TForm
{
__published:    // Компоненты, управляемые средой IDE.
    TLabel *JobsLabel;
    TStatusBar *StatusBar;
private:        // Объявления пользователя.
    void __fastcall WMSpoolerStatus(TWMSpoolerStatus &message);
public:         // Объявления пользователя.
    __fastcall TMainForm(TComponent* Owner);

BEGIN_MESSAGE_MAP
    MESSAGE_HANDLER(WM_SPOOLERSTATUS,
                    TWMSpoolerStatus,
                    WMSpoolerStatus)
END_MESSAGE_MAP(TForm)
} ;
```

Метод WMSpoolerStatus() в вашем исходном файле должен быть подобен представленному в листинге 23.22.

Листинг 23.22. Обработчик сообщения WM_SPOOLERSTATUS

```
void __fastcall TMainForm::WMSpoolerStatus(TWMSpoolerStatus &msg)
{
    // Чтение данных из сообщения.
    int iStatus     = msg.JobStatus;
    int iNumOfJobs  = msg.JobsLeft;

    // Запись количества заданий печати в имена, помеченные как JobsLabel.
    JobsLabel->Caption = "В очереди заданий осталось: "+
                         IntToStr(iNumOfJobs);

    // Обновляем текст в строке состояния для уведомления о том,
    // что сообщение получено.
    StatusBar->SimpleText = "Сообщение получено...";

    // Передаем сообщение, чтобы другие также могли его получить.
    msg.Result = false;
}
```

Если вы заинтересованы в перехвате заданий печати и работе с содержимым этих данных, то лучше всего самостоятельно написать пользовательский процессор печати на базе примера программы, содержащегося в пакетах драйверов устройств (Device Driver Kit — DDK) Windows 9x/NT/2000. К сожалению, вам придется написать процессор печати для каждой версии Windows, т.е. один вариант для Windows 95/98/98 Second Edition, второй — для Windows NT 4 и еще один — для Windows 2000. В интерфейсе Win32 API предусмотрены следующие функции для инсталляции, удаления и настройки процессора печати.

- Инсталляция. Функция GetPrintProcessorDirectory() возвращает папку Windows, в которой должен быть инсталлирован процессор печати. Инсталляцию процессора печати выполняет функция AddPrintProcessor().
- Удаление. Функция DeletePrintProcessor() удаляет процессор печати из системы.
- Настройка. Чтобы указать имя процессора печати для принтера, используйте поле pPrintProcessor в структуре PRINTER_INFO_2. Для задания типа данных, с которыми будет работать принтер, используйте поле pDataType (наиболее употребительны такие типы, как RAW, EMF и TEXT). Для получения списка инсталлированных в системе процессоров печати используйте функцию EnumPrintProcessors().

Как перехватить нажатие кнопки <Print Screen>

Есть два способа, позволяющие перехватить нажатие кнопки <Print Screen>.

Первый — состоит в создании метода обработки сообщений, который для перехвата сообщения VK_SNAPSHOT использует функцию GetAsyncKeyState() (для вызова этого метода установлено событие OnIdle() класса Tapplication). Приведенный ниже фрагмент программы взят из примера CatchPrintScreen_1, который находится на прилагаемом к книге компакт-диске.

Внесите следующее объявление в класс вашей формы (в заголовочный файл).

```
void __fastcall AppIdle(TObject *Sender, bool &Done);
```

Внесите следующий код в событие OnCreate() формы (в заголовочный файл).

```
void __fastcall TMainForm::FormCreate(TObject *Sender)
{
    Application->OnIdle = AppIdle;
}
```

Затем реализуйте функцию AppIdle:

```
void __fastcall TMainForm::AppIdle(TObject *Sender, bool &Done)
{
    if (GetAsyncKeyState(VK_SNAPSHOT) != 0)
        Application->MessageBox("Got it!", "Print Screen Caught",
                                MB_OK | MB_ICONINFORMATION);
    Done = True;
}
```

Второй способ заключается в создании отображения для перехвата сообщения WM_HOTKEY (его VCL-инкапсуляция называется TWMHotKey), как показано в листинге 23.21 для сообщения WM_SPOOLERSTATUS. Зарегистрируйте функциональную клавишу с помощью функции RegisterHotKey() в событии формы OnCreate() и отмените регистрацию, используя функцию UnregisterHotKey() в обработчике события формы OnDestroy(). Заголовочный файл вашей формы должен иметь следующий вид (взято из примера CatchPrintScreen_2, который находится на прилагаемом к книге компакт-диске).

```
const int ID_SNAPSHOT = 1000;

class TMainForm : public TForm
{
__published:    // Компоненты, управляемые средой IDE.
    void __fastcall FormCreate(TObject *Sender);
    void __fastcall FormDestroy(TObject *Sender);
private:        // Объявления пользователя.
```

```
    void __fastcall WMHotKey(TWMHotKey &msg);
public:             // Объявления пользователя.
    __fastcall TMainForm(TComponent* Owner);

BEGIN_MESSAGE_MAP
    MESSAGE_HANDLER(WM_HOTKEY,TWMHotKey,WMHotKey)
END_MESSAGE_MAP(TForm)
} ;
```

Ваш исходный файл должен включать следующий код.

```
void __fastcall TMainForm::FormCreate(TObject *Sender)
{
    // ID_SNAPSHOT объявляется как константа в заголовочном файле для формы.
    RegisterHotKey(Handle, ID_SNAPSHOT, 0, VK_SNAPSHOT);
}
//---------------------------------------------------------------
void __fastcall TMainForm::FormDestroy(TObject *Sender)
{
    UnregisterHotKey(Handle, ID_SNAPSHOT);
}
//---------------------------------------------------------------
void __fastcall TMainForm::WMHotKey(TWMHotKey &msg)
{
    if (msg.HotKey == ID_SNAPSHOT)
    {
        Application->MessageBox("Got it!", "Print Screen Caught",
                                MB_OK | MB_ICONINFORMATION);
        // Передаем сообщение, чтобы его могли получить и другие.
        msg.Result = false;
    }
}
```

Печать формы

В VCL-классе `TForm` определен метод `Print()`, который выводит на действующий по умолчанию принтер видимые части окна. Масштабирование печати зависит от свойства формы `PrintScale`, которое может иметь значения `poNone`, `poProportional` и `poPrintToFit`. Функция формы `Print()` имеет много недостатков — например, распечатываются только видимые части формы. Поэтому лучше для печати формы использовать класс `TExcellentFormPrinter`, который распространяется условно-бесплатно и его можно загрузить из Internet (`http://www.code4sale.com/joehecht`).

Создание компонента предварительного просмотра печати

Чтобы подробно разъяснить, как создать собственный компонент предварительного просмотра печати, потребовалась бы целая глава. Однако следующий пример может подсказать идею решения этой задачи. И хотя его трудно назвать коммерческим продуктом, из него можно почерпнуть несколько советов по использованию функций `SetWindowExtEx()` и `SetViewportExtEx()`, позволяющих выполнить надлежащее преобразование между координа-

тами экрана и принтера. Источник предварительно просматриваемых перед печатью данных реализован как класс TBitmap, чего как раз не следует делать в реальном компоненте предварительного просмотра печати по причине большого количества присущих ему ограничений. Лучше всего использовать класс TMetaFile. Проект, о котором идет речь, находится на прилагаемом к книге компакт-диске и называется PrintPreview.

Использование процедур преобразования для вывода данных на принтер

Следующие функции, примененные к печати текста и растровых изображений, способны облегчить работу с такими единицами измерения длины, как миллиметры или дюймы. Процедуры, приведенные в листинге 23.23, для управления преобразованиями между пикселями и реальными единицами измерения используют свойство MeasureUnit (типа TMeasureUnit), возможными значениями которого являются Millimeters и Inches. iDefaultDPI — это целочисленная константа со значением 300, которая используется в случаях неудачного обращения к функции GetDeviceCaps(). Она позволяет выходить "сухим из воды", т.е. служит "ремнем безопасности" при аварийных ситуациях.

Листинг 23.23. Процедуры преобразования единиц измерения принтера

```
enum TMeasureUnit {  Inches, Millimeters } ;

float __fastcall PixelsToMeasureUnitHorz(unsigned int Pixels)
{
    // Локальная переменная.
    float fTmp;

    try
    {
        // функция GetDeviceCaps возвращает пиксели на дюйм.
        fTmp = (float)Pixels/GetDeviceCaps(Printer()->Handle,
                                           LOGPIXELSX);
    }
    catch(...)
    {
        // Устанавливаем значение, измеренное при
        // стандартном разрешении (300).
        fTmp = (float)Pixels/iDefaultDPI;
    }

    // Преобразуем его в миллиметры или возвращаем в дюймах.
    if (MeasureUnit == Millimeters)
        // Преобразуем из дюймов в миллиметры.
        return fTmp*25.4;
    else
        return fTmp;

}
//---------------------------------------------------------------
float __fastcall PixelsToMeasureUnitVert (unsigned int Pixels)
{
```

```
    // Локальная переменная.
    float fTmp;

    try
{
        // Функция GetDeviceCaps возвращает пиксели на дюйм.
        fTmp = (float)Pixels/GetDeviceCaps(Printer()->Handle,
                                           LOGPIXELSY);
    }
    catch(...)
    {
        // Устанавливаем значение, измеренное при
        // стандартном разрешении (300).
        fTmp = (float)Pixels/iDefaultDPI;
    }

    // Преобразуем его в миллиметры или возвращаем в дюймах.
    if (MeasureUnit == Millimeters)
        // Преобразуем из дюймов в миллиметры.
        return fTmp*25.4;
    else
        return fTmp;

}
//---------------------------------------------------------------
unsigned int __fastcall MeasureUnitToPixelsHorz(float Measure)
{
    // Локальная переменная.
    float fTmp;

    try
    {
        // Функция GetDeviceCaps возвращает пиксели на дюйм.
        fTmp = (float)Measure * GetDeviceCaps(Printer()->Handle,
                                              LOGPIXELSX);
    }
    catch(...)
    {
        // Устанавливаем значение, измеренное при
        // стандартном разрешении (300).
        fTmp = (float)Measure * iDefaultDPI;
    }

    // Какое преобразование в пиксели мы выполняем: из
    // миллиметров или дюймов?
    if (MeasureUnit == Millimeters)
        // Преобразуем из дюймов в миллиметры.
        return (int)fTmp/25.4;
    else
        return (int)fTmp;
}
```

```
//---------------------------------------------------------------
unsigned int __fastcall MeasureUnitToPixelsVert(float Measure)
{
    // Локальная переменная.
    float fTmp;

    try
    {
        // Функция GetDeviceCaps возвращает пиксели на дюйм.
        fTmp = (float)Measure * GetDeviceCaps(Printer()->Handle,
                                              LOGPIXELSY);
    }
    catch(...)
    {
        // Устанавливаем значение, измеренное при
        // стандартном разрешении (300).
        fTmp = (float)Measure * iDefaultDPI;
    }

    // Какое преобразование в пиксели мы выполняем: из
    // миллиметров или дюймов?
    if (MeasureUnit == Millimeters)
        // Преобразуем из дюймов в миллиметры.
        return (int)fTmp/25.4;
    else
        return (int)fTmp;
}
```

Другая информация, связанная с выводом данных на принтер

Ниже приводится список распространенных вопросов о печати и ответы на них. Если вы хотите найти ответ на вопрос, отсутствующий в этом списке, просмотрите содержимое Web-узлов DejaNews (`www.deja.com`) или MSDN (`http://msdn.microsoft.com`).

Как отобразить диалоговое окно установки принтера?

Вызовите функцию `OpenPrinter()`, чтобы получить дескриптор принтера, для которого вы собираетесь отобразить диалоговое окно Принтер (Printer Setup). Затем вызовите функцию `PrinterProperties()` с дескриптором вашей формы и дескриптором принтера, только что полученным с помощью функции `OpenPrinter()`. Наконец, не забудьте выполнить функцию `ClosePrinter()` с использованием дескриптора вашего принтера.

Как добавить в систему новый принтер?

Используйте функцию `AddPrinter()` и задайте параметры принтера, передав их с помощью заранее заполненной структуры `PRINTER_INFO_2`. Для обеспечения работоспособности функции `AddPrinter()` необходимо позаботиться об инсталляции драйвера добавляемого принтера. Для получения списка всех инсталлированных в системе драйверов принтеров используйте функцию `EnumPrinterDrivers()`. В одной "связке" с функцией `AddPrinter()` работает функция `DeletePrinter()`.

Как инсталлировать драйвер нового принтера?

Для инсталляции драйвера принтера используйте функцию `AddPrinterDriver()`, а чтобы узнать имя папки, в которой должен быть размещен драйвер перед инсталляцией, воспользуйтесь функцией `GetPrinterDriverDirectory()`. В одной "связке" с функцией `AddPrinterDriver()` работает функция `DeletePrinterDriver()`.

Как создать соединение с принтером?

Для создания соединения с принтером используйте функцию `AddPrinterConnection()` или вызовите функцию `ConnectToPrinterDlg()`, чтобы открыть диалоговое окно, позволяющее пользователям делать обзор дерева каталогов и подключаться к принтерам по сети. Обязательным "партнером" функции `AddPrinterConnection()` является функция `DeletePrinterConnection()`.

Создание диаграмм с помощью компонента Tchart

Компонентом создания диаграмм `TChart` от компании Steema Software SL оснащены среды как Delphi, так C++Builder (с версии 3). Он представляет собой очень мощную библиотеку процедур построения диаграмм, написанных с использованием VCL, и поставляется в трех вариантах.

- Стандартная версия — компонент `TChart`, расположенный во вкладке Additional (Дополнительно) палитры компонентов (Component Palette).
- Версия работы с данными — компонент `TDBChart`, расположенный во вкладке Data Controls (Элементы управления данными) палитры компонентов.
- Версия QuickReport — компонент `TQRChart`, расположенный во вкладке QReport палитры компонентов.

Мы остановимся на стандартной версии.

Основной пакет характеристик стандартной версии компонента `TChart` составляют 11 типов рядов, 7 статистических функций, а также редактирование диаграмм и рядов, построенных на стадии проектирования. Он включает огромный массив объектов, которые в данном случае являются отдельными частями диаграммы: оси, ряды, заголовки, легенды и пр.

Версия, включенная в поставку C++Builder, содержит много примеров приложений и надлежащую документацию, но пример программы в справочной системе написан не на C++, а на Delphi. Это создает определенные трудности для желающих разобраться в более сложных методах построения диаграмм, поэтому здесь будут рассмотрены в общих чертах некоторые из этих методов. Web-узел `TChart` (`www.steema.com`) содержит широкую подборку часто задаваемых вопросов (FAQ) и советов как по C++Builder, так и Delphi.

Начнем с мастера TeeChart

Нет ничего проще, чем добавить диаграмму к приложению. Для этого достаточно опустить компонент `TChart` в форму и при необходимости изменить размеры диаграммы. Компонент `TChart` выведен из класса `TPanel` и поэтому наследует его свойства.

На заметку

Диаграмму также можно добавить с помощью диалогового окна New Application (Создать приложение). Выберите команду File⇨New (Файл⇨Создать) и дважды щелкните на пиктограмме TeeChart Wizard, расположенной во вкладке Business (Бизнес).

Чтобы добавить в диаграмму один или несколько рядов и изменить громадный массив свойств диаграммы и ряда, достаточно дважды щелкнуть на этой диаграмме, запустив тем самым редактор диаграмм.

Редактирование диаграмм

Редактор диаграмм и рядов предоставляет доступ к большинству свойств диаграммы и ряда, причем более интуитивно понятным образом, чем Object Inspector. Этот редактор имеет две основные вкладки: Chart (Диаграмма), которая предоставляет доступ к свойствам диаграмм и упрощает добавление ряда к диаграмме, и Series (Ряд), которая позволяет изменять свойства ряда (рис. 23.4).

При добавлении в диаграмму нового ряда происходит автоматическое заполнение этого ряда случайными данными (только во время разработки), что позволяет сразу же увидеть результат изменения свойств диаграммы и ряда.

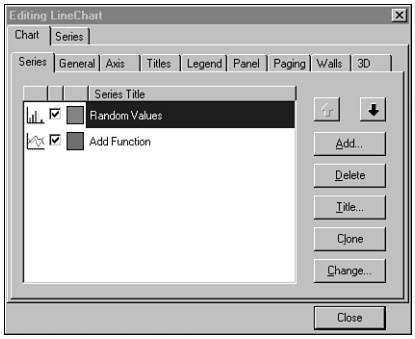


Рис. 23.4. Редактор диаграмм и рядов

Добавление данных в диаграмму

Добавить в диаграмму данные — дело нехитрое; для этого достаточно использовать метод `Add()`. При вызове этого метода в ряд данных добавляется только одно значение, поэтому для добавления массива значений следует организовать цикл, подобный следующему.

```
for (int i=0; i<TotalPointCount; i++)
{
    Series1->Add(SomeData[i], (String) (i+1), clTeeColor);
}
```

Метод `Add()` имеет три аргумента. Первый содержит добавляемое значение (в данном случае элемент массива); второй представляет собой подпись для соответствующей точки диаграммы, а третий означает ее цвет (константа `clTeeColor` присваивает данному значению цвет, используемый по умолчанию).

Очевидно, каждая точка на диаграмме имеет значение `x` (позиция точки на оси "x", или горизонтальной оси) и значение `y` (позиция точки на оси "y", или вертикальной оси). Метод `Add()` автоматически присваивает `x`-значения ряда, поэтому каждое последующее значение в массиве `y` (в данном примере это массив `SomeData`) получит последующее значение `x`. Следовательно, значению `y` с нулевым индексом в массиве будет соответствовать нулевое значение `x`, значению `y` с индексом 1 в массиве будет соответствовать значение `x`, равное 1, и т.д.

Чтобы задать значение `x` для каждого значения `y` и, следовательно, создать настоящий график рассеивания, используйте метод `AddXY()`. Форма этой функции совпадает с формой метода `Add()` за исключением дополнительного аргумента, который представляет собой массив значений `x`.

```
for (int i=0; i<TotalPointCount; i++)
{
    Series1->AddXY(XData[i], YData[i], (String) (i+1),
                                                clTeeColor);
}
```

Можно одним махом присвоить массив значений всему ряду диаграммы, избежав возни с добавлением по одному значению. На Web-узле TeeChart можно познакомиться с очень удачным примером использования этого метода, который сопровождается подробными разъяснениями, поэтому мы опустим здесь его описание.

Изменение внешнего вида диаграмм при выполнении приложений

Внешний вид диаграммы, "приобретенный" в период разработки с помощью редактора свойств диаграмм и рядов, часто подходит лишь для очень простых приложений, работающих с диаграммами. Однако благодаря возможности изменить свойства диаграмм и рядов во время работы приложений, можно сделать диаграммы более интересными для пользователей и улучшить визуальное представление данных.

Проект `Chart.bpr`, который находится на компакт-диске, прилагаемом к этой книге, демонстрирует некоторые полезные методы компонента `TChart` (они подробно описаны в следующих разделах).

Как упростить процесс изменения свойств

Неудивительно, что все публикуемые свойства диаграмм и рядов можно изменить во время работы приложения. Это очень просто делается и не требует длинных пояснений. Упомянутое выше приложение работы с диаграммой позволяет пользователю переходить от двухмерного (2D) к трехмерному (3D) изображению диаграмм (и обратно) с помощью специального флажка. Код, изменяющий это свойство диаграмм, находится в обработчике события `OnClick` этого флажка.

```
LineChart->View3D = ViewCheck->Checked;
```

Диаграммы, построенные на основе компонента `TChart`, обновляются (перерисовываются) автоматически при изменении их свойств, поэтому от пользователя никаких дополнительных действий не требуется.

Использование события `OnGetBarStyle`

Большинство объектов диаграмм также реагируют на события. К этим событиям обычно можно получить доступ с помощью C++Builder-средства Object Inspector, но некоторые из них не публикуются и, следовательно, не отражены в документации (либо весьма скупо). Возможно, самым полезным из них является событие `OnGetBarStyle`, которое происходит каждый раз, когда рисуется отдельный элемент в ряду элементов. Это позволяет менять стиль или цвет используемого элемента в зависимости от его индекса, что и продемонстрировано в примере приложения работы с диаграммой.

Сначала нужно назначить ряду (элементов) обработчик событий. В данной программе ряд диаграммы называется `Series1`, а обработчик событий назначается в конструкторе формы.

```
Series1->OnGetBarStyle = this->GetBarStyle;
```

Затем нужно определить функцию обработчика событий (как показано ниже). Аргументами этой функции являются: `Sender` типа `TCustomBarSeries` (указатель на ряд элементов, который вызвал это событие), `ValueIndex` (индекс элемента, который должен быть нарисован) и `BarStyle` (ссылка на переменную типа `TBarStyle`). Последний аргумент и позволяет менять стиль изображаемого элемента диаграммы.

```
void __fastcall TChartForm::GetBarStyle(TCustomBarSeries *Sender,
                                        int ValueIndex, TBarStyle &BarStyle)
{
    TBarStyle Style[6] = { bsArrow,
    bsCilinder,
    bsEllipse,
    bsInvPyramid,
    bsPyramid,
    bsRectangle} ;

    BarStyle = Style[(ValueIndex + 1) % 6];
}
```

На заметку Обратите внимание на странное написание значения `bsCilinder` типа `TBarStyle`. Это не опечатка!

Функция в этом примере просто присваивает стиль каждому элементу на основе его позиции в ряду и изменяет его цвет (это не показано).

Работа с диаграммами

Помимо возможности создания проекта диаграммы в период разработки приложения и изменения ее в процессе работы, компонент `TChart` обеспечивает механизмы, которые позволяют пользователю динамически взаимодействовать с диаграммами. Это достигается с помощью обработчиков событий и функций, предусмотренных для объекта диаграммы. В результате перед пользователем открываются большие возможности, позволяющие, например, считывать y-значения диаграммы, изменять цвета, корректировать масштаб осей и добавлять или удалять элементы ряда.

Получение значений диаграммы

Используя событие `OnClickSeries`, нетрудно прочитать значение столбца в столбцовой диаграмме (гистограмме). Это событие имеет следующие аргументы: `Sender` типа `TCustomChart` (идентифицирует диаграмму, запускающую это событие), `Series` типа `TChartSeries` (идентифицирует ряд, для которого произошло это событие), `ValueIndex` (предоставляет индекс элемента, на котором был выполнен щелчок мышью) и обычные аргументы, связанные с координатами мыши.

```
void __fastcall TChartForm::LineChartClickSeries(
                                                TCustomChart *Sender,
                                                TChartSeries *Series,
                                                int ValueIndex,
                                                TMouseButton Button,
                                                TShiftState Shift, int X, int Y)
{
}
```

Каждый объект класса `TChartSeries` содержит список значений ряда, к которым можно получить доступ с помощью свойства `YValues->Value`.

```
YValue->Caption = Series->YValueToText(Value);
```

В приведенной выше строке программы это значение преобразуется в текст и отображается в главной форме приложения в виде подписи.

Преобразование экранных координат в координаты диаграммы

Ненамного труднее преобразовать экранные координаты в координаты диаграммы. Это даст возможность написать функции, которые, например, позволят пользователю создать прямоугольник выделения вокруг различных точек на диаграмме и выполнять над ними необходимые действия.

В следующем примере программы этот метод используется для отображения координат `x` и `y` диаграммы в заголовке формы, причем это происходит всякий раз, когда указатель мыши располагается над диаграммой. Используя событие диаграммы `OnMouseMove` (вспомните, что класс `TChart` выведен из класса `TPanel`), мы должны сначала прочитать экранные координаты ограничительного прямоугольника диаграммы.

```
TChart *TheChart = dynamic_cast<TChart>(Sender);
TRect TheChartRect = TheChart->ChartRect;
```

Экранные координаты можно преобразовать в координаты диаграммы только в том случае, когда указатель мыши располагается над областью диаграммы (в пределах ее осей), поэтому необходимо убедиться, что указатель мыши действительно находится внутри этой области.

```
if ((X > TheChartRect.Left && X < TheChartRect.Right)
                  && (Y > TheChartRect.Top && Y < TheChartRect.Bottom))
{
}
```

Здесь `X` и `Y` — экранные координаты указателя мыши, переданные обработчику события `OnMouseMove`.

Для преобразования экранных координат в координаты диаграммы предусмотрена функция `CalcPosPoint()` function, которая является членом компонента `TChartAxis`, и любая конкретная диаграмма, вероятно, будет иметь по крайней мере два связанных с ней объекта этого класса: один для горизонтальной оси, или оси X, и один для вертикальной оси, или оси Y. Таким образом, для преобразования экранной координаты позиции мыши `Y` в координату `Y` диаграммы достаточно выполнить следующую строку программы.

```
double YValue = TheChart->LeftAxis->CalcPosPoint(Y);
```

С компонентом `TChartAxis` связано множество полезных функций и свойств, позволяющих управлять осями и настраивать внешний вид диаграммы.

Динамическое создание диаграмм

В предыдущих разделах шла речь о действиях, осуществляемых как динамически, так и статически (т.е. в период разработки) над диаграммами, созданными при разработке приложения. В этом разделе описываются методы динамического построения диаграмм (во время работы приложения), которые использованы в примере программы для создания 20 круговых диаграмм в прокручиваемой области.

Поскольку предполагается построение нескольких диаграмм, прежде всего необходимо объявить массив диаграмм и рядов диаграмм. В данном примере программы это делается в конструкторе формы.

```
PieCharts = new TChart *[PieChartCount];
PieSeries = new TPieSeries *[PieChartCount];
```

Обязательно позаботьтесь о корректном объявлении указателей на массивы (в данном примере это `PieCharts` и `PieSeries`) в заголовочном файле формы. Поскольку мы создаем массив VCL-объектов, они должны быть объявлены как указатели на указатели.

```
TChart **PieCharts;
TPieSeries **PieSeries;
```

Затем нужно назначить форме каждую диаграмму и каждый ряд диаграммы и сделать их родителем область прокрутки.

```
PieCharts[i] = new TChart(this);
PieSeries[i] = new TPieSeries(this);
PieCharts[i]->Parent = PieChartScrollBox;
```

Затем каждый ряд нужно присвоить диаграмме, поскольку объект диаграммы не имеет типа ряда, автоматически связанного с ним. Теперь вам должно быть понятно, как присвоить диаграмме несколько рядов (может быть, даже разного типа):

```
PieCharts[i]->AddSeries(PieSeries[i]);
```

Наконец, нам осталось лишь изменить некоторые свойства диаграмм и присвоить им данные, как описано выше.

Печать диаграмм

Существует ряд способов вывода диаграмм на печать. Самый простой из них — использовать метод `Print()` формы, содержащей диаграмму. Этот метод прекрасно зарекомендовал себя при выполнении контрольных распечаток. Однако качество воспроизведения диаграмм с его помощью оставляет желать лучшего.

К счастью, в компоненте `TChart` предусмотрено несколько собственных методов печати, которые генерируют диаграммы гораздо более высокого качества, чем метод печати формы. Самым простым из них является метод `Print()`.

```
LineChart->Print();
```

При выполнении этой строки программы диаграмма будет напечатана с использованием полных значений ширины и высоты печатной страницы при разрешении экрана, действующего по умолчанию. Однако печать при таком разрешении экрана часто не походит для приложений, включающих диаграммы (в конце концов, разрешение принтера намного больше чем у экрана). К счастью, можно изменить это разрешение, а следовательно, и качество печатаемых диаграмм. Каждый компонент `TChart` имеет динамическое свойство `PrintResolution`, которое можно установить равным целому значению в диапазоне –100–0, где –100 означает, что диаграмма печатается при максимальном разрешении принтера, а 0 — при установленном разрешении экрана.

```
LineChart->PrintResolution = PrintResScroll->Position;
```

На заметку В статье справочного файла компонента `TChart`, посвященной свойству `PrintResolution`, указано, что оно предназначено только для чтения. Это неверно!

Используя методы `PrintPartial` и `PrintRect`, можно также задать прямоугольник для диаграммы с целью ее масштабирования на печатной странице.

Использование метода PrintPartialCanvas

С помощью метода PrintPartialCanvas на одной странице можно напечатать сразу несколько диаграмм. Метод позволяет вывести одну или несколько диаграмм прямо на канву принтера и, в отличие от описанных методов печати, автоматически не запускает задание печати и не выталкивает напечатанную страницу.

Чтобы подготовить несколько диаграмм для печати на одной странице, сначала для каждой диаграммы определите объект класса TRect, т.е. прямоугольник, который описывает положение диаграммы на странице. Например, следующие строки программы описывают область, которая соответствует верхней трети печатной страницы при полной ее ширине.

```
PrintRect[0].Left = 0;
PrintRect[0].Right = (Printer()->PageWidth - 1);
PrintRect[0].Top = 0;
PrintRect[0].Bottom = (Printer()->PageHeight - 1) / 3;
```

Затем запустите задание, но позаботьтесь о том, чтобы для каждой диаграммы сначала было установлено разрешение на печать.

```
Printer()->BeginDoc();
```

Для всех диаграмм, подлежащих печати на одной странице, всегда устанавливайте разрешение до открытия страницы принтера с помощью функции Printer()->BeginDoc().

Затем для каждой диаграммы, печатаемой на одной странице (для которой прямоугольник уже определен), вызовите метод PrintPartialCanvas(). В качестве аргументов он принимает элемент типа TCanvas и элемент типа TRect. Элемент типа TCanvas представляет собой канву принтера, а элемент типа Trect — прямоугольник, определенный для диаграммы, печатаемой в данный момент.

```
LineChart->PrintPartialCanvas(Printer()->Canvas, PrintRect[0]);
```

Вызвав метод PrintPartialCanvas для каждой печатаемой диаграммы, печатаем страницу.

```
Printer()->EndDoc();
```

При печати таким способом существует небольшая проблема. Время от времени с размерами текста и графических изображений на печатаемой странице начинает происходить что-то неладное, и их приходится восстанавливать вручную. Если эта проблема возникнет при печати диаграмм в приложении, нужно до вывода на принтер получить режим исходного отображения контекста устройства принтера.

```
int MapMode = GetMapMode(Printer()->Handle);
```

После того как все экземпляры будут посланы на канву принтера, восстанавливаем режим отображения.

```
SetMapMode(Printer()->Handle, MapMode);
```

Такой метод “лечения” поможет при решении любых проблем масштабирования, возникающих во время печати.

Переход к TeeChart Pro

Стандартная версия мастера TeeChart, которая поставляется с C++Builder и Delphi, несомненно, обладает богатыми возможностями и вполне подходит для построения отдельных диаграмм. Но для регулярного использования в качестве средства построения диаграмм стоит

перейти к версии TeeChart Pro, которая включает возможности стандартной версии и позволяет использовать некоторый набор дополнительных, а именно:

- 100%-й исходный код;
- 9 дополнительных типов рядов;
- 6 пользовательских типов рядов;
- 9 дополнительных статистических функций;
- интерфейс OpenGL 3D;
- динамический доступ к редактору диаграмм и рядов.

Один из дополнительных типов ряда, в сочетании с использованием интерфейса OpenGL 3D, позволяет строить трехмерные диаграммы поверхности (т.е. они имеют три оси — `x`, `y` и `z`). Однако самым существенным новшеством, возможно, является динамический доступ к редактору диаграмм. Поэтому пользователь может строить диаграммы во время выполнения приложения, и даже расширять возможности этого редактора, например, добавляя свои кнопки.

Существует также ActiveX-версия средства TeeChart, которая способна строить диаграммы в режиме реального времени для работы с ними в Internet.

Резюме

В этой главе мы рассмотрели создание средства просмотра отчетов QuickReport, которое позволяет просматривать отчеты, не печатая их. Мы исследовали методы сохранения и загрузки отчетов с целью совместной работы с ними различных пользователей. Кроме того, здесь были описаны как базовые, так и более сложные методы печати текста и графических изображений с помощью API- и VCL-функций. В круг рассмотренных тем вошли такие "вечные" проблемы, как непосредственный вывод текста на принтер, печать растровых изображений, чтение и запись свойств принтера и вопросы масштабирования.

Наконец, мы кратко описали возможности построения диаграмм на основе компонента `TChart`, включая динамическое их создание, редактирование и печать.

Глава

24

Использование интерфейса Win32 API

Пол Густавсон
Крис Винтерс

Среда разработки Borland C++Builder предназначена специально для создания 32-разрядных приложений Windows. Ключевым словом здесь является *Windows*, но необходимо уточнить, что сказанное относится к таким версиям, как Microsoft Windows 9x, Millennium (Me), NT и 2000. Со временем Borland Kylix (предполагаемая среда разработки для операционной системы Linux) будет поддерживать и создание 32-разрядных приложений Linux, но пока основное внимание C++Builder направлено на Windows. C++Builder упрощает и ускоряет процесс создания 32-разрядных приложений Windows, использующих язык C++. Для облегчения разработки Windows-приложений C++Builder позволяет разработчику получать доступ и активно использовать 32-разрядный *интерфейс программирования приложений* (Application Programming Interface — API) Windows (Win32), причем как прямо, так и косвенно, т.е. через компоненты программного обеспечения промежуточного уровня. В этой главе рассматривается интерфейс Win32 API "в чистом виде", а также некоторые примеры его использования.

Win32 API в сравнении с программными средствами Win32 промежуточного уровня

Компанией Borland поддерживается несколько пакетов ресурсов, которые облегчают разработку Windows-приложений и способствуют сокращению общих затрат на нее. В контексте разработки программных продуктов такой пакет обеспечивает структуру, интерфейс, а зачастую и строительные блоки, которыми может воспользоваться разработчик в процессе создания своей программы. Самым значительным пакетом, знакомым разработчикам, имеющим дело с продуктами Borland, является *библиотека визуальных компонентов* (Visual Component Library — VCL). Варианты VCL подробно описаны в главах 8 и 9. VCL считается ключевым ингредиентом в Borland-средах ускоренной разработки приложений (Rapid Application Development — RAD) Delphi и C++Builder, т.е. VCL делает RAD-технологию реальностью. До появления VCL компания Borland разработала библиотеку объектов Windows (Object Windows Library — OWL), пакет для создания C++-приложений, предназначенный для более ранних версий Windows. Borland также поддерживает библиотеку базовых классов Microsoft (Microsoft Foundation Class (MFC) Library), может быть, самый популярный пакет для создания C++-приложений Windows. Будь то библиотека VCL, OWL или MFC, компоненты промежуточного уровня или пакеты — все они предназначены для облегчения разработки посредством инкапсуляции многих функций 32-разрядного Windows-интерфейса (Win32) программирования приложений (API) в объектах и методах многократного использования.

В большинстве случаев неплохо было бы по крайней мере рассмотреть роль Win32-интерфейса промежуточного уровня в разработке приложений Windows. Прежде всего промежуточный уровень сокращает временны*е* и человеческие затраты на написание Windows-программ. Более того, такие компоненты промежуточного уровня, как библиотеки VCL, OWL и MFC, обеспечивают объектно-ориентированный интерфейс и оболочку для более структурированного интерфейса Win32 API. Несмотря на эти преимущества, существует множество случаев, когда стоит использовать исходные функции Win32 API непосредственно в программе. Дело в том, что "голая" функция Win32 API позволяет увеличить быстродействие и сократить расходы памяти, по сравнению с функциями промежуточного уровня. Поскольку промежуточный уровень по сути является абстрактным интерфейсом и оболочкой для интерфейса Win32 API, его производительность может несколько пострадать из-за множества параметрических условий, которые необходимо поддерживать. Более того, разработчики, использующие функции Win32 API, предпочитают иметь более полное представление о том, как их приложения взаимодействуют с операционной системой, другими приложениями и периферийными устройствами. Но компоненты промежуточного уровня намеренно маски-

руют связь между приложением и операционной системой (operating system — OS). К счастью, исходный код многих компонентов VCL распространяется в составе профессиональной (Professional) и промышленной (Enterprise) версий пакета C++Builder и большинством библиотек VCL сторонних производителей. Исследование исходного кода помогает разработчику понять сложные элементы функций Win32 API, входящие в состав некоторых классов компонентов или базовых классов.

Возможно, самым неопровержимым доводом для разработчиков C++Builder в пользу вызова "голых" функций Win32 прямо из приложения является ситуация (причем вполне вероятная), когда нужное действие не обеспечено библиотекой VCL. Опытные программисты часто сталкиваются с тем, что VCL бессильна удовлетворить их потребности. В этом случае интерфейс Win32 API просто "затыкает дыру", обеспечивая программисту требуемую функциональность и гибкость. Поэтому благодаря объектно-ориентированной природе интерфейса и возможности многократного использования VCL, в тех случаях, когда компонент VCL способен обеспечить нужную функциональность, настоятельно рекомендуется использовать именно его, причем столько раз, сколько это необходимо. Однако если не существует компонента VCL, способного выполнить поставленную задачу, можно прибегнуть к услугам Win32 API.

Краткая историческая справка по Windows и API

Чтобы полностью охватить широкие возможности Win32 API, полезно оглянуться назад, на историю возникновения OS Windows и интерфейса Windows API. В начале 1980-х годов компания Microsoft приступила к разработке продукта, именуемого Interface Manager, который позже стал известен как Windows. По общему мнению, Microsoft, а также Apple, кредитовали разработки на базе работ, начатых компанией Xerox Palo Alto Research Center (PARC). Xerox PARC представила идею среды графических окон, больше известной сегодня как графический интерфейс пользователя (Graphical User Interface — GUI). Цель Microsoft (была и остается) — обеспечить легко используемый GUI-интерфейс с независимой от устройств графикой и поддержкой многозадачной обработки. Кстати, фирма Apple развивала аналогичную идею, известную под названием *Lisa*, но она не заявила о себе, пока мир не увидел систему Macintosh (той же фирмы Apple) и не почувствовал потенциал, заложенный в современной GUI-среде.

В 1985 году компания Microsoft выпустила на суд общественности версию Windows 1.0. Она была предназначена для расширения системы DOS за счет обеспечения графического интерфейса для приложений, хотя по своему внешнему виду это была еще "DOS-овская операционка". Безо всяких способностей к перекрытию приложения отныне можно было "накладывать" одно на другое. В пределах одного и того же пространства памяти теперь могло работать сразу несколько приложений. На рынке PC подобные возможности предлагались и другими фирмами-производителями, например Quarterdeck (DESQview), IBM (TopView) и Digital Research (GEM). Уникальный аспект каждой из этих сред состоял в том, что пользователь мог запускать более одного приложения, и благодаря вмешательству со стороны пользователя (нажатие клавиши) среда могла "переключать задачи" между приложениями. Переключение задач позволяло не только активизировать одно из выполняемых приложений, но создавало впечатление параллельной работы сразу нескольких приложений.

В Windows 1.0 переключение задач между приложениями было реализовано на основе функции API `GetMessage()`. API-функция `GlobalAlloc()` часто использовалась для выделения памяти приложениям в пределах доступного пространства памяти. Обеими

этими API-функциями по-прежнему можно пользоваться, хотя с тех пор они были модифицированы и усовершенствованы. Все же их редко используют для разработки новых Windows-приложений.

В 1987 году была выпущена версия Windows 2.0, которая приобрела знакомое многим "лицо" с пиктограммами и перекрывающимися окнами. Интерфейс Win API предоставил улучшенную поддержку создания диалоговых окон и работы с ними. DOS-приложения можно было запускать под управлением Windows посредством PIF-файлов. Windows 2.0 наконец преодолела 1-мегабайтовый барьер памяти, который служил помехой для ранних версий DOS. Версию Windows 2.0 заменила улучшенная версия 2.1, но вследствие появления чипа Intel 80386, следующая версия Windows, которая была направлена на поддержку новых возможностей архитектуры Intel, вышла под названием Win/386. Впоследствии был выпущен еще один вариант версии Windows 2.1 в виде Win/286, чтобы привести в соответствие эти два продукта с текущими архитектурами Intel. В частности, версия Win/386 использовала преимущества нового процессора Intel путем обеспечения возможности многозадачной работы без вытеснения, что значительно улучшало схему многозадачной работы, реализованной в версии 1.0, которая требовала использования функции `GetMessage()`. Путем выделения каждому приложению определенного интервала времени центрального процессора (central processing unit — CPU) задача переключения между приложениями могла быть выполнена практически автономно.

В 1990 году вышла в свет версия Windows 3.0, и мир медленно стал поворачиваться лицом к этим продуктам. Версия Windows 3.0 (Win 3.0) обеспечила поддержку до 16 Мбайт оперативной памяти, или ОЗУ (Random Access Memory — RAM), с использованием расширенного режима 386-го процессора. В интерфейсе API была реализована улучшенная поддержка графики и управления памятью. Функцию `GlobalAlloc()` можно было теперь использовать для выделения из кучи областей памяти большего объема, и она позволяла смещать адресацию памяти приложения на основе сегментированной памяти. Вместе с Windows стали передавать пользователю такие приложения-утилиты, как Program Manager (диспетчер программ) и File Manager (диспетчер файлов). Получили распространение различные надстройки и OS-расширения, а также библиотеки поддержки; например, мультимедийные расширения до сих пор поддерживаются библиотекой динамической компоновки (dynamic link library — DLL) Windows Multimedia System (MMSYSTEM).

Выпущенная в апреле 1992 года версия Windows 3.1 (помимо мультимедийных возможностей в виде функции `PlaySound()`) обеспечила поддержку шрифтов TrueType и технологии связывания и встраивания объектов (object linking and embedding — OLE). Были предусмотрены общие диалоговые окна, которые упростили для программистов разработку таких обычных задач, как печать, сохранение и открытие файлов, а также выбор цветов. Изготовители ПК начали продавать свои системы в комплекте с OS Windows 3.1. На этом этапе Windows обеспечила поддержку только для адресного пространства 16-разрядной сегментированной памяти и 16-разрядных целых, несмотря на тот факт, что 386-й и 486-й процессоры Intel разработаны с учетом 32-разрядной архитектуры. В начале 1994 года компания Microsoft выпустила Windows for Workgroups (WFW), версия 3.11. WFW обеспечила недорогую *сетевую* операционную систему и стала неотъемлемым элементом на корпоративных рабочих столах Америки. Она также предназначена для интеграции с OS-ориентированными системами Windows New Technology (NT) (о них речь впереди). Интерфейс API, доступный для Win 3.x (включая WFW), содержит законченный набор функций, структур и сообщений обратного вызова, известных в наши дни как интерфейс Win16 API.

В 1993 году, как раз перед выходом версии WFW, компания Microsoft как-то без лишнего шума выпустила Windows New Technology (NT) версии 3.1, хотя NT впервые предоставила первый настоящий API-пакет, ориентированный на 32-разрядную архитектурную модель Windows. В ядре операционной системы были использованы преимущества 486-го процессо-

ра Intel, разработанного с учетом 32-разрядной архитектуры, посредством обеспечения настоящей многозадачной системы с вытеснением и реализована поддержка 32-разрядного плоского (простого) адресного пространства и 32-разрядных целых. Вскоре после выхода в свет NT был выпущен интерфейс Win32s API, который сделал возможным для разработчиков создание 32-разрядных приложений для Windows 3.x. 32-разрядная поддержка была реализована в Windows 3.x с помощью библиотеки динамической компоновки (DLL), которая действовала "по доверенности" путем преобразования 32-разрядных функций в 16-разрядные, требуемые ядром. Таким образом, "корни" этой OS по-прежнему были в 16-разрядном сегментированном адресном пространстве и ограничивались поддержкой 16-разрядных целых. Однако интерфейс Win32s предоставил программистам возможность начать проектирование и разработку приложений для таких 32-разрядных систем Windows, как NT и последующей Windows 95.

В августе 1995 года "под фанфары" была выпущена в свет Windows 95. Интерфейс пользователя этой операционной системы был полностью обновлен и усовершенствован, а сама OS была укомплектована тем же вариантом интерфейса Win32 API, который использовался в Windows NT 3.51. Подобно NT, Windows 95 обеспечивала поддержку параллельных процессов, независимых потоков выполнения и плоскую (линейную) 32-разрядную адресную модель, но ей недоставало многих средств обеспечения безопасности и серверных возможностей, реализованных в NT. Кроме того, она еще была связана с дисковой операционной системой (Disk Operation System — DOS), хотя эта "порочная" связь была несколько замаскированной. Однако самым большим ее козырем было то, что большинство Win 3.x-приложений, разработанных с использованием Win16 API и Win32s API, могли безупречно выполняться под управлением Windows 95. Этот аспект был весьма привлекателен для потребителей.

Высказывались критические замечания, суть которых состояла в том, что большинство существующих Win 3.x-приложений были просто несовместимы с NT, а NT недоставало значительной части репертуара Win32-приложений. На состоявшейся в Орландо в июне 1994 года (за год до выхода в свет Windows 95) конференции Borland Developer's Conference был проведен опрос слушателей направления Borland C++ Product Address, чтобы определить количество программистов, разрабатывающих приложения для различных операционных сред. Бо́льшая часть присутствующих (несколько сотен человек) поднятием рук сообщила, что разрабатывает приложения для Win 3.x. Около двадцати разработчиков засвидетельствовали в пользу OS/2 (IBM). Однако лишь два человека из нескольких сотен присутствующих дали понять, что они выполняли разработки приложений для NT. Причина в том, что не было потребности в приложениях для платформы NT. Известно, что Windows 95 поддерживала разработку Win32 и при этом не обесценила Win16-приложения, и только одного этого было достаточно для того, чтобы приложения начали переводить на платформу Win32 и NT. По общему мнению, Win95 создавалась как временный механизм для стимулирования разработок 32-разрядных Windows-приложений и для перевода потребителей полностью на "рельсы" NT. Для Win95 изначально готовилась судьба "временно исполняющей обязанности" OS, чтобы в конце концов ее заменила потребительская версия NT. Первый шаг в этом направлении был сделан, когда компания Microsoft обновила NT, сделав ее аналогичной Windows 95. NT версии 4.0 была выпущена в августе 1996 года и принята корпоративной Америкой "на ура". Однако ей недоставало мультимедийной поддержки, чего очень жаждали многие потребители.

В конечном счете Microsoft обновила Windows 95 для поддержки универсальной последовательной шины (Universal Serial Bus) и 32-разрядной таблицы размещения файлов (File Allocation Table — FAT), выпустив в 1997 году версию Windows 95 OEM (Original Equipment Manufacturer) Service Release 2 (OSR2). Версия Win95 OSR2 характеризуется дополнительными функциями, которыми пополнился интерфейс Win32 API, включающий элементы Web-

направленности, используемые броузером Internet Explorer версии 3.0, новые и обновленные средства графической и мультимедийной поддержки (обеспечиваемые посредством таких продуктов, как DirectX 2, Open GL 1.1 и ActiveMovie), а также другие элементы (например, интерфейс Cryptographic API и новая функция `GetDiskFreeSpaceEx()`, которая обеспечивает поддержку жестких дисковых устройств большего объема). Выпущенная в июне 1998 года версия Windows 98 содержала около 10 000 API-функций. Не нужно говорить о том, что Microsoft постоянно добавляет в Windows функции API посредством обновлений и новых версий.

После выхода Windows 98 компания Microsoft выпустила в мае 1999 года версию Windows 98 Second Edition (SE), в феврале 2000 года — Windows 2000 и в сентябре 2000 года — Windows Millenium (Me). Windows 2000 заменила линию продуктов NT 4.0, но еще не вытеснила ряд Win 9x. В Windows Me содержались обновления линии продуктов 98 — отказ от экрана начальной загрузки DOS, новые мультимедийные возможности и прикладные инструменты для пользователей, но все они опирались на архитектуру Win 9x. Как Windows 98/Me, так и Windows 2000 фундаментально отличаются на уровне ядра, но большую часть текущей версии интерфейса Win32 API поддерживают Win 9x, Me, NT и 2000. В результате приложения, разработанные для Windows 98/Me, будут нормально работать под управлением Windows 2000. И наоборот, за некоторыми исключениями, приложения Windows 2000 также работают в среде Windows 98/Me. В этой неудержимой динамике интерфейс Win API не подвергся каким бы то ни было кардинальным изменениям, но продолжает постоянно улучшаться и расширяться. С середины 80-х годов, когда для Windows 1.0 был представлен первый вариант, интерфейс Win API значительно вырос, но даже в сегодняшнем API остается много оригинальных процедур и функций обратного вызова. Под влиянием на компьютерную индустрию новых технологий введенный компанией Microsoft интерфейс API будет постоянно подвергаться пересмотрам и обновлениям. Несмотря на новые “веяния”, большинство текущих структур и процедур API будут по-прежнему поддерживаться последующими версиями Windows. Безусловно, со временем будут обнародованы (а значит, станут использоваться) ранее недокументированные или неопубликованные функции Win32 API. Успех Windows был и будет зависеть от доступности интерфейса Win32 API. Знание Win32 API и надлежащий уровень его использования позволяет разработчикам создавать высококачественные приложения и компоненты, следствием чего является возрастающий спрос на них.

Функциональные области Win32 API

Этот краткий курс истории, возможно, “прольет свет” на интерфейс Windows API, и вы поймете, для поддержки чего он предназначен. Win32 API содержит функции, функции обратного вызова (callback) и структуры, необходимые для поддержки интеграции, управления и работы приложения в операционной среде Windows. Функции API часто предоставляют Windows-приложению общее диалоговое окно для открытия и сохранения файлов или отправки документов на принтер. Существует и ряд других задач, которые также успешно решает интерфейс Win32 API. В средах Windows 98, Me и Windows 2000 доступно более 10 000 элементов API. Основные системные библиотеки Windows, которые обычно находятся в папке `Windows/System`, содержат большинство этих процедур Win32 API. В число ключевых входят такие библиотеки, как библиотека ядра Windows (`kernel32.dll`), библиотека пользователя (`user32.dll`) и библиотека графического интерфейса с устройствами (graphical device interface — GDI) (`gdi32.dll`).

Кроме того, важное значение имеют некоторые расширения и вспомогательные библиотеки, представленные Microsoft (и другими фирмами-производителями) за период эволюции

Windows. Например, когда Windows 3.x и Windows 95 заявили о себе как операционные среды будущего, компания Microsoft еще не представляла, насколько сильно "всемирная паутина" (World Wide Web) повлияет на роль Windows. Борьба между Web-броузером компании Netscape и Microsoft оказала сильнейшее воздействие на расширения Win32 API и вспомогательные библиотеки "родом" из Microsoft.

В число расширений и вспомогательных библиотек входят: библиотека SHELL и библиотека MMSYSTEM, Windows Sockets (WinSock), DirectX и Windows Internet Extensions (WinInet) API. Расширения и библиотеки поддержки не являются обязательным элементом для работы Windows, но они обеспечивают для разработчиков набор средств, которые делают их программы более ошибкоустойчивыми и мощными. Для получения полного списка доступных функций и услуг, предоставляемых любой из библиотек Win32, достаточно использовать вызываемую из командной строки программу Borland `impdef`, чтобы создать файл определения интерфейса (`.def`) библиотеки DLL, как показано в следующем примере.

```
impdef -a user32.def user32.dll
```

Файл с расширением `.def` (его можно просмотреть в любом текстовом редакторе) будет содержать список доступных функций, которые могут использовать разработчики приложений. Если вы заглянете в папку `windows/system`, то обнаружите там огромное количество DLL-файлов, при этом лишь небольшая их часть окажется настоящими DLL-библиотеками Win32 API. Обычно в имя файла каждой библиотеки Win32 DLL входит число `32`. В диалоговом окне Properties (Свойства) (рис. 24.1) программы Windows Explorer (Проводник Windows) можно найти больше информации о DLL. Если вы увидите там слово *Microsoft* и аббревиатуру *API*, то, вероятнее всего, это и есть Win32 API.

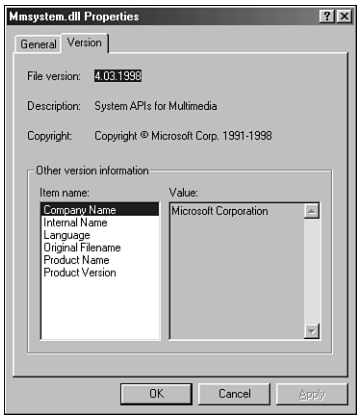


Рис. 24.1. Диалоговое окно Properties

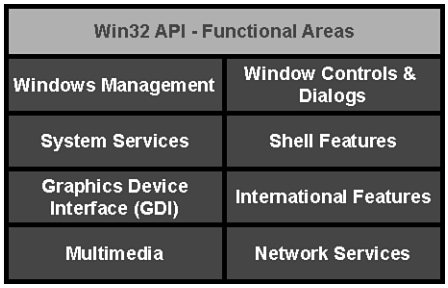


Рис. 24.2. Структурная схема функциональных областей Win32 API

Попытка понять каждый элемент интерфейса API — практически невыполнимая задача, которая под силу только очень сильным личностям, поскольку при каждом обновлении Windows или Internet Explorer добавляются новые функции API, многие из которых недокументированны. Однако основные функциональные области Windows, как показано на рис. 24.2, позволяют понять композицию интерфейса API и возможности, предоставляемые им для разработчиков.

Интерфейс Win32 API можно разделить на восемь функциональных областей.

- Управление окнами (Windows Management)

- Системные службы (System Services)
- Графический интерфейс с устройствами (Graphical Device Interface)
- Мультимедийные службы (Multimedia Services)
- Общие элементы управления и диалоговые окна (Common Controls and Dialogs)
- Функции оболочки (Shell Features)
- Межнациональные средства (International Features)
- Сетевые службы (Network Services)

Рассмотрим каждую из этих областей в отдельности.

Управление окнами

Создание и управление приложениями Windows реализуется благодаря функциям, поддерживаемым библиотекой пользователя (`user32.dll`). Интерфейс библиотеки пользователя включает возможности по управлению окнами и меню, диалоговыми окнами, окнами сообщений, доступом к мыши и клавиатуре, а также другими встроенными элементами.

Важно понимать саму идею окна. *Окно* действует как интерфейс между пользователем и приложением. Приложением Windows создается по крайней мере одно окно, именуемое главным. Приложения вольны, конечно, создавать также дополнительные окна. Основная цель окна — принимать от пользователя входную и отображать выходную информацию. Функции категории *Windows Management* используются для обеспечения возможности создания окон и использования их приложениями. Входные данные от мыши и клавиатуры принимаются главным окном посредством сообщений, которые передаются системой с помощью функций категории управления окнами. Функции этой категории также позволяют приложениям отображать пиктограммы, меню и диалоговые окна, которые принимают и отображают дополнительную информацию пользователя.

В табл. 24.1 приведены наиболее популярные API-функции категории Windows Management, используемые для создания и управления окнами. К этим функциям доступ осуществляется путем включения в файл исходного кода приложения заголовочного файла `windows.h` или заголовочного файла Borland `vcl.h`.

Таблица 24.1. Распространенные функции управления окнами

Функции управления окнами	Описание
`CascadeWindows()`	Располагает каскадом указанные окна или дочерние окна указанного родительского окна
`CloseWindow()`	Сворачивает, но не разрушает указанное окно
`CreateWindow()`	Создает перекрываемое, всплывающее (контекстное) или дочернее окно
`DestroyWindow()`	Разрушает окно. Система отвечает отправлением сообщения `WM_DESTROY` заданному окну
`EnableWindow()`	Разрешает или запрещает ввод данных от мыши или клавиатуры в указанное окно или элемент управления
`EnumWindows()`	Регистрирует посредством циклического опроса все отображаемые окна верхнего уровня с последующей передачей дескриптора отдельно каждого окна функции обратного вызова, определенной приложением

Окончание табл. 24.1

Функции управления окнами	Описание
`EnumWindowsProc()`	Используется функцией `EnumWindows()`. Это определенная приложением функция обратного вызова, которую функция `EnumWindows()` использует для передачи дескрипторов окнам верхнего уровня
`FindWindow()`	Считывает дескриптор для окна верхнего уровня, имя класса и имя окна которого совпадают с заданными строками
`FindWindowEx()`	Считывает дескрипторы доступных активных окон. Аналогична функции `FindWindow()`, но также обеспечивает поддержку для обнаружения дочерних окон
`GetWindowRect()`	Считывает координаты экрана заданного окна
`GetWindowText()`	Считывает содержимое строки заголовка указанного окна
`MoveWindow()`	Изменяет местоположение и размеры указанного окна
`SetWindowText()`	Модифицирует текст строки заголовка для указанного окна
`ShowWindow()`	Устанавливает состояние отображения указанного окна. Возможны такие состояния отображения, как скрытие, разворачивание, сворачивание, восстановление и активизация окна
`TileWindows()`	Располагает указанные окна или дочерние окна указанного родительского окна по черепичному способу (без перекрытий)
`WinMain()`	Вызывается системой в качестве начальной точки входа для Win32-ориентированного приложения

За рамками этой таблицы осталось множество других функций управления окнами. В действительности в версии 4.10.2222 библиотеки `user32.dll` насчитывается в общей сложности 648 функций. Для просмотра полного списка доступных функций категории Windows Management используйте описанную выше программу Borland `impdef`, вызываемую из командной строки.

Системные службы

Функции системного сервиса позволяют приложению управлять и контролировать ресурсы, предоставлять доступ к файлам, папкам, а также входным и выходным устройствам. Кроме того, они разрешают приложению регистрировать события и обрабатывать ошибки и исключительные ситуации. Более того, функции этой категории обеспечивают средства, которые можно использовать для создания приложений других типов, например консольных приложений и драйверных служб.

Библиотека ядра Windows (`kernel32.dll`) содержит большинство системных служебных функций низкого уровня для операционной среды. Они обеспечивают доступ к файлам, управление памятью и ресурсами, а также поддержку многозадачной и многопотоковой обработки. Все приложения Windows используют для этого ядро Windows. Например, когда приложению нужна память (как при запуске, так и во время работы), оно требует, чтобы ядро Windows выделило необходимую область памяти.

Ключевые аспекты системного сервиса описаны в табл. 24.2.

Таблица 24.2. Основные системные службы

Подкатегория	Описание
Неделимость	Поддержка совместного использования строк приложениями посредством 16-разрядных целочисленных идентификаторов
Связь	Поддержка таких средств связи, как последовательные порты, параллельные порты и модемы
Поддержка консоли	Поддержка управления вводом/выводом для приложений, работающих в символьном (а не GUI-) режиме
Отладка	Обеспечивает поддержку обработки событий для отладки приложений
Устройства ввода/вывода	Поддержка связи драйверов устройств в приложении
Динамический обмен данными (Dynamic Data Exchange — DDE)	Поддержка передачи данных между приложениями. Функции DDE в настоящее время поддерживаются библиотекой пользователя (`user32.dll`) и библиотекой управления динамическим обменом данными (Dynamic Data Exchange Management Library — DDEML)
Библиотека динамической компоновки (Dynamic Link Library — DLL)	Поддержка создания библиотек, которые могут загружаться приложением во время выполнения, и управления ими
Сообщения об ошибках	Поддержка обработки сообщений, например с помощью функций `MessageBeep()` и `FlashWindow()`
Регистрация событий	Поддержка записи событий приложения в журнал регистрации
Отображение файлов	Поддержка отображения содержимого файла на виртуальное адресное пространство
Файлы	Поддержка ввода/вывода файлов на запоминающие устройства
Дескрипторы и объекты	Поддержка создания и управления дескрипторами и объектами, благодаря чему обеспечивается абстрактный и безопасный доступ к системным ресурсам Windows
Поддержка справочной системы	Поддержка процедур, используемых в сочетании со справочным приложением Windows (Windows Help). Поддержка справочной системы на самом деле осуществляется библиотекой пользователя (`user32.dll`), но компания Microsoft считает ее составной частью системных служб
Взаимодействие процессов (Interprocess Communication — IPC)	Поддержка механизмов, облегчающих связь и совместное использование данных между приложениями. Эти механизмы включают отображение файлов, общую память, анонимные и именованные каналы, почтовые каналы (mailslots) и системный буфер обмена (Clipboard)
Операции над целыми большой длины	Поддержка операций над 64-разрядными целыми
Почтовые каналы (mailslots)	Поддержка создания односторонних IPC-каналов (mailslots) и управление ими
Управление памятью	Поддержка выделения и использования памяти

Окончание табл. 24.2

Подкатегория	Описание
Каналы	Поддержка создания, управления и использования каналов. Каналы — это IPC-соединения, позволяющие одному процессу взаимодействовать с другим
Обработка файлов в формате Portable Executable	Поддержка обработки или доступа к файлу в формате Portable Executable (PE) посредством процедур, предоставляемых библиотекой `imagehlp.dll`. PE — это формат загружаемого кода, используемый сервисом подкачки компонентов из Internet (Internet Component Download). Файл в формате PE представляет собой одиночный исполняемый файл с расширением типа OCX, DLL или EXE
Управление электропитанием	Содержит функции и сообщения, которые отображают состояние системы электропитания и уведомляют о событиях, связанных с управлением электропитанием
Процессы и потоки	Поддержка многозадачности и системы планировщика задач. Создание и управление несколькими потоками и дочерними процессами внутри приложения
Системный реестр	Поддержка хранения, доступа и управления системной базой данных с помощью данных конфигурации приложения и системных компонентов
Безопасность	Поддержка доступа со стороны приложения или пользователя к некоторому объекту либо отказ в таковом. Многие процедуры безопасности содержатся в библиотеке Advanced API (`advapi32.dll`)
Службы	Поддержка автоматизированных служб, в которых приложение (или драйвер) может действовать без участия пользователя (или знаний пользователя). Поддержкой приложений таких типов управляет Менеджер служб (Service Control Manager — SCM). Многие служебные функции содержатся в библиотеке Advanced API (`advapi32.dll`)
Манипуляции над строками	Поддержка копирования, сравнения, сортировки, форматирования и преобразования символьных строк, а также определения типов символов
Структурированная обработка исключительных ситуаций	Обеспечивает поддержку компилятора при обработке исключительных ситуаций и прекращении действий
Синхронизация	Обеспечивает механизмы, которые могут использовать потоки для синхронизации доступа к ресурсам
Системная информация	Поддержка определения и считывания такой системной информации, как имя компьютера, имя пользователя, значения переменных среды, тип процессора и данные о системных цветах
Системные сообщения	Поддержка уведомления приложений и драйверов устройств о событиях
Выключение системы	Поддержка выхода из системы текущего пользователя или выключения системы
Стриммер (Tape Backup)	Поддержка разрешения, выдаваемого приложениям резервирования, выполнять операции чтения/записи на магнитную ленту, а также инициализации и считывания информации о накопителе на магнитной ленте
Время	Поддержка чтения и записи даты и времени для системы, файлов и местного часового пояса
Windows-станции и рабочий стол	Поддержка служб Win32 для вызова функций USER32 и GDI32 независимо от сеанса учетной записи, в котором эта служба работает

В текущей версии Windows доступно в общей сложности 745 функций библиотеки ядра системы (`kernel32.dll`). Они охватывают подкатегории, описанные в табл. 24.2. Для просмотра полного списка доступных системных функций, предоставляемых библиотекой ядра, используйте программу Borland `impdef`, вызываемую из командной строки (как описано выше). Имейте также в виду, что существуют и другие служебные функции, которые выполняют системную поддержку, но хранятся в библиотеке пользователя и некоторых других служебных библиотеках (например, `imagehlp.dll` и `advapi32.dll`).

Графический интерфейс с устройствами

Графический интерфейс с устройствами (graphical device interface — GDI), поддерживаемый библиотекой динамической компоновки `gdi32.dll`, предоставляет возможность окну выполнять операции рисования и печати. Этой подкатегорией функций охватываются такие направления, как рисование линий, операции с текстом, шрифтом и управление цветом.

Одним из ключевых элементов GDI является контекст устройства. *Контекст устройства* (device context — DC) представляет структуру данных, определяющую набор графических объектов, атрибутов и режимов работы. Контексты устройств создаются с помощью функций `CreateDC()` и `GetDC()`. Существует множество других DC-функций, которые используются не менее часто. Различают семь типов GDI-объектов, которые можно выбрать в контексте устройства (табл. 24.3).

Таблица 24.3. Типы GDI-объектов

Тип объекта	Описание
Растр (bitmap)	Используется для копирования или прокрутки частей экрана
Кисть (brush)	Используется для окрашивания или заполнения многоугольников, эллипсов и контуров
Шрифт (font)	Используется для идентификации типа, размера и стиля (начертания) шрифта
Палитра (palette)	Используется для определения набора доступных цветов
Контур (path)	Используется в операциях окрашивания и рисования
Перо (pen)	Используется для рисования линий
Область (region)	Используется для отсечения и других операций

В текущей версии Windows доступно в общей сложности 334 GDI-функции. Для просмотра полного списка доступных GDI-функций, предоставляемых Windows-библиотекой `gdi32.dll`, используйте вызываемую из командной строки программу Borland `impdef`.

Windows-интерфейс GDI может оказаться чрезвычайно полезным для проведения двумерной графической обработки и визуализации бизнес-приложений. В библиотеке VCL компании Borland богатыми функциональными GDI-возможностями наделены классы `TImage` и `TCanvas`. Поскольку VCL обеспечивает солидную инкапсуляцию GDI-интерфейса, то для разработчика C++Builder нет большого резона в использовании Win32 GDI в "натуральном виде".

Слабым место интерфейса GDI (причем неважно, в каком виде "употребляемого": напрямую или посредством VCL) является его низкая производительность. Использование GDI для отображения быстроменяющихся графических образов реального времени часто разочаровывает как разработчиков, так и пользователей. Даже при наличии отличного оборудования его скорость (в кадрах в секунду) остается очень низкой. К счастью, интерфейс DirectDraw API, который является частью такого комплекса инструментальных средств Microsoft, как DirectX Game SDK, предлагает более приемлемую библиотечную альтернативу для работы с двумерной графикой. DirectDraw — просто находка для тех, кому нужна высокопроизводительная

двумерная визуализация. Хотя DirectX считается расширением интерфейса Win32 API (его поддерживает как Windows 9x, так и NT 2000), он заслуживает отдельной главы. Поэтому за дополнительной информацией по DirectX обращайтесь к главе 26.

Мультимедийные службы

Все больше приложений включают такие мультимедийные элементы, как звук и видео. В состав некоторых дополнительных библиотек Microsoft для Windows входят Multimedia System Library (`mmsystem.dll`), Microsoft Video for Windows Library (`msvfw32.dll`) и Microsoft Audio Compression Manager Library (`msacm32.dll`). Первоначально мультимедийные средства не являлись частью интерфейса Windows API. Однако со временем операционная система Windows "дозрела" до их поддержки. Платформа Windows отвечает практически всем мультимедийным потребностям пользователя и соперничает с возможностями телевидения и стереозвучания. Windows поддерживает мультимедийные устройства и такие форматы данных, как MIDI (Musical Instrument Digital Interface — цифровой интерфейс музыкальных инструментов), а также аудио- и видеосигналы. Большая часть основных мультимедийных возможностей обеспечивается библиотеками `MMSYSTEM.DLL`, `MSVFW32.DLL` и `MSACM32.DLL`. Для более качественной поддержки игр, музыки и видео компания Microsoft также разработала мультимедийный интерфейс API, известный как DirectX. Однако в этом разделе основное внимание уделено возможностям, предоставляемым библиотеками `MMSYSTEM.DLL`, `MSVFW32.DLL` и `MSACM32.DLL`. Эти DLL используют такие мультимедийные заголовочные файлы, как `DIGITALV.H`, `MCIAVI.H`, `MMSYSTEM.H`, `MSACM.H`, `VCR.H` и `VFW.H`.

Основные мультимедийные услуги, предоставляемые этими библиотеками, описаны в табл. 24.4.

Таблица 24.4. Мультимедийные службы

Подкатегория	Описание
Менеджер сжатия аудиосигналов (Audio Compression Manager — ACM)	Обеспечивает системную поддержку сжатия, распаковки, фильтрации и преобразования аудиосигналов (использует `MSACM32.DLL`)
Звуковые микшеры (Audio Mixers)	Предоставляет услуги по управлению маршрутизацией аудиолиний к приемному устройству для воспроизведения или записи. Обеспечивает также регуляцию громкости и создания других эффектов (использует `MMSYSTEM.DLL`)
AVICap	Предоставляет возможности по "захвату" видеоизображений, включающие поддержку интерфейса для сбора данных с видео- и аудиоаппаратуры, а также поддержку направления видеоданных на диск (использует `MSVFW32.DLL`)
AVIFile	Обеспечивает функции и макросы для доступа к файлам в формате AVI (*a*udio-*v*ideo *i*nterleaved files), содержащим перемежающиеся аудио- и видеоданные (использует `MSVFW32.DLL`)
DrawDib	Обеспечивает GDI-независимые функции, используемые для передачи независимых от устройств растровых изображений (*d*evice *i*ndependent *b*itmaps — DIB) в видеопамять (использует `MSVFW32.DLL`)
Джойстики (Joysticks)	Обеспечивает поддержку управления джойстиками и другими вспомогательными устройствами ввода, которые отслеживают позиции внутри некоторой абсолютной координатной системы (сенсорный экран, графический планшет и световое перо) (использует `MMSYSTEM.DLL`)

Окончание табл. 24.4

Подкатегория	Описание
Класс окон MCIWnd	Предоставляет класс окон для управления мультимедийными устройствами. Обеспечивает поддержку простого добавления в любое приложение возможностей воспроизведения или записи данных мультимедийного характера (использует `MSVFW32.DLL`)
Интерфейс управления средой передачи информации (Media Control Interface — MCI)	Стандартный управляющий интерфейс для мультимедийных устройств и файлов обеспечивает независимую от устройств поддержку устройств мультимедийного воспроизведения (устройства восстановления волновой формы сигналов, CD-плэйер, цифровой музыкальный синтезатор и цифровые видеоустройства) и записи файлов мультимедийных ресурсов (использует `MMSYSTEM.DLL`)
Ввод/вывод мультимедийных файлов (Multimedia File I/O)	Предоставляет услуги по буферизированному и небуферизированному вводу-выводу файлов и обеспечивает поддержку файлов в формате RIFF (Resource Interchange File Format — формат файлов для обмена ресурсами), т.е. аудио- и видеофайлов для одновременного получения "живого" видеоизображения, текста и звукового сопровождения (использует `MMSYSTEM.DLL`)
Мультимедийные таймеры (Multimedia Timers)	Обеспечивает поддержку планирования периодического возникновения событий таймера с высокой разрешающей способностью (использует `MMSYSTEM.DLL`)
Цифровой интерфейс музыкальных инструментов (Music Instrument Digital Interface — MIDI)	Обеспечивает MIDI-преобразователь для перевода и перенаправления поступающих MIDI-сообщений и другие виды MIDI-сервиса (запрос устройств и управление ими, распределение и запись данных MIDI-сообщений). Примечание: MCI-сервис, обеспечивающий работу цифрового музыкального синтезатора, можно сочетать с MIDI-сервисом. (Использует `MMSYSTEM.DLL`)
Менеджер сжатия видеоизображения (Video Compression Manager — VCM)	Обеспечивает сжатие и распаковку видеоданных (использует `MSVFW32.DLL`)
Аудиосигналы (Waveform-Audio)	Предоставляет утилиты для добавления (воспроизведения и записи) аудиосигналов (использует `MMSYSTEM.DLL`)

На три упомянутые в табл. 24.4 библиотеки DLL приходится более 240 функций, которые поддерживают работу мультимедийных служб. Для просмотра полного списка доступных мультимедийных функций (содержащихся в `MMSYSTEM.DLL`, `MSVFW32.DLL` и `MSACM32.DLL`) используйте описанную выше программу Borland `impdef`, вызываемую из командной строки. Библиотеками импорта для этих DLL служат `WINMM.LIB`, `VFW32.LIB` и `MSACM32.LIB`, соответственно.

Общие элементы управления и диалоговые окна

Коллекция встроенных элементов управления обязана своим существованием в среде Windows библиотеке `COMCTL32.DLL`, а коллекция общих диалоговых окон — библиотеке `COMDLG32.DLL`. Возможность использования общих элементов управления и диалоговых окон позволяет разработчикам быстро применить в своей программе общие элементы интерфейса вместо того чтобы тратить время и силы на написание собственных оконных интерфейсов. В

большинстве случаев пользователям достаточно использовать палитру C++Builder с элементами управления и диалоговыми окнами, обеспечиваемую библиотекой визуальных компонентов (Visual Component Library — VCL). Однако важно, чтобы разработчики знали о общих элементах управления и диалоговых окнах, к которым можно получить доступ с помощью интерфейса Win32 API.

Существует 22 общих элемента управления для Windows в версии 4.71 библиотеки `COMCTL32.DLL`, которая поставлялась в составе Internet Explorer версии 4.0. При рассмотрении заголовочного файла `commctrl.h`, входящего в поставку C++Builder, можно узнать, какие элементы Win32 доступны с помощью библиотеки общих элементов управления. Эти элементы управления описаны в табл. 24.5.

Таблица 24.5. Общие элементы управления

Элемент управления	Описание	Эквивалент Borland VCL
Элемент управления анимацией	Воспроизводит простые видеоклипы (AVI-файл) без звука	`TAnimate`
ComboBoxEx*	Обеспечивает поддержку для изображений элементов внутри комбинированного списка	Нет
Элемент сбора информации о дате и времени (Date and Time Picker)	Предоставляет интерфейс для обмена с пользователем информацией о дате и времени	`TDateTimePicker`
Список с перетаскиваемыми элементами (Drag List Box)	Предоставляет окно списка, позволяющее перетаскивать элементы из одной позиции в другую	Нет
Полоса прокрутки (Flat Scroll Bar*)	Предоставляет уникальную трехмерную полосу прокрутки, создаваемую с помощью функции `InitializeFlatSB()`	Нет
Элементы управления заголовками (Header Controls)	Используется для размещения заголовков изменяемого размера над столбцами чисел или текста	`THeaderControl`
Элементы управления "горячими клавишами" (Hot-Key Controls)	Позволяет ввести комбинацию "горячих клавиш"	`THotKey`
Список изображений (Image List)	Предоставляет коллекцию изображений одинакового размера, адресуемых по индексу. Эта коллекция создается с помощью функции `ImageListCreate()`	`TImageList`
Адрес, определяемый протоколом IP, или IP-адрес (IP Address*)	Позволяет ввести IP-адрес в понятном для пользователя формате	Нет
Просмотр списка (List View)	Отображает коллекцию элементов в виде больших или маленьких значков, детализированной таблицы или с возможностью просмотра содержимого в специальном окне	`TListView`
Помесячный календарь (Monthly Calendar)	Предоставляет графический интерфейс с календарем для просмотра и выбора дат	`TMonthCalendar`

Окончание табл. 24.5

Элемент управления	Описание	Эквивалент Borland VCL
Прокручиваемое окно (Page Scroller*)	Предоставляет прокручиваемое окно, содержащее дочернее окно (в виде элемента управления), которое слишком велико, чтобы его можно было видеть целиком	`TPageScroller`
Индикаторы выполнения (Progress Bars)	Используются для графического представления степени выполнения длительных операций	`TProgressBar`
Окна свойств (Property Sheets)	Позволяют просматривать и редактировать свойства элемента в форме с вкладками	`TPageControl`
Комбинированная полоса (Rebar (Coolbar))	Используется для включения одной или нескольких полос, представляющих собой комбинацию элемента захвата (gripper bar), растрового изображения, текстовой надписи и дочернего окна	`TCoolBar`
Строка состояния (Status Bar)	Предоставляет панель, используемую для отображения текстовой и графической информации	`TStatusBar`
Вкладки (Tab Controls)	Определяет несколько вкладок внутри области окна (по аналогии с алфавитными разделителями в записной книжке)	`TTabControl`
Панели инструментов (Toolbars)	Содержит на панели одну или несколько кнопок, которые можно выбрать	`TToolBar`
Всплывающие подсказки (Tooltip Controls)	Используется для отображения небольшого по размеру всплывающего окна, отображающего текстовое описание	`THintWindow`
Ползунковые индикаторы (Trackbars)	Предоставляет ползунковый индикатор с необязательными отметками, используемыми для подбора целого значения в пределах заданного диапазона	`TTrackBar`
Средства просмотра дерева (Tree View)	Отображает в рамках окна иерархический список элементов и подэлементов, состоящих из надписи и необязательного растрового изображения	`TTreeView`
Счетчики (Up-Down Controls)	Предоставляет элемент редактирования, состоящий из пары кнопок со стрелками, используемыми для увеличения или уменьшения значения	`TUpDown`

* Введено в Internet Explorer 4.0.

Эти элементы управления (если не указан иной вариант) создаются с помощью функции `InitCommonControlsEx()` либо функции `CreateWindowEx()` посредством выбора соответствующих признаков (флагов). Большинство из них обеспечивается библиотекой Borland VCL. В библиотеке общих элементов управления (`comctl32.dll`) доступно 82 функции, которые поддерживают эти и другие элементы управления. Для просмотра полного списка доступных

Win32 API-функций поддержки общих элементов управления используйте описанную выше программу Borland impdef, вызываемую из командной строки.

Помимо элементов управления, Microsoft обеспечивает и поддержку диалоговых окон. Версия 4.00.950 библиотеки COMDLG32.DLL содержит восемь общих диалоговых окон. Однако C++Builder для каждого из этих окон предоставляет VCL-оболочки, как показано в табл. 24.6.

Таблица 24.6. VCL-эквиваленты для Microsoft API-функций создания и поддержки общих диалоговых окон

Диалоговое окно	Функция Microsoft API	VCL-эквивалент
Choose Color (Выбор цвета)	ChooseColor()	TColorDialog
Choose Font (Выбор шрифта)	ChooseFont()	TFontDialog
Find Text (Найти текст)	FindText()	TFindDialog
Open File (Открытие файла)	GetOpenNameFile()	TOpenDialog
Save File (Сохранение файла)	GetSaveFileName()	TSaveDialog
Print Setup (Параметры страницы)	PageSetupDlg()	TPrinterSetupDialog
Print (Печать)	PrintDlg()	TPrintDialog
Replace Text (Заменить текст)	ReplaceText()	TReplaceDialog

Рассмотрение заголовочного файла commdlg.h, поставляемого вместе с C++Builder, позволит ознакомиться с Win32-элементами, доступными с помощью библиотеки общих диалоговых окон, а создание .def-файла для библиотеки commdlg.dll поможет получить полное представление о доступных API-функциях.

Функции оболочки

Термин *оболочка* (shell) используется в Windows для описания приложения, которое позволяет пользователю группировать, запускать и управлять другими приложениями. В число возможностей оболочки входит поддержка средства “перетащить и опустить”, поддержка связи файлов для запуска и поиска других приложений, а также способность выделения пиктограмм из других файлов. Этот мощный набор средств Win32 API содержится в библиотеке Shell32.dll. В состав любого приложения, реализующего поведение оболочки, обязательно должен входить заголовочный файл SHELLAPI.H. Основные возможности Shell API описаны в табл. 24.7.

Таблица 24.7. Основные возможности интерфейса Shell API

Средство	Описание
Перетащить и опустить	Позволяет пользователю выбрать один или несколько файлов в Windows Explorer (даже в старой версии File Explorer, входящей в состав Win 3.x), которые можно перетащить и опустить в любое открытое приложение, где предварительно была использована функция оболочки DragAcceptFiles. Открытым приложением принимается сообщение WM_DROPFILES, которое используется для считывания имен файлов и отображения позиции, в которую опускаются файлы. Это реализуется посредством функций оболочки DragQueryFile и DragQueryPoint

Окончание табл. 24.7

Средство	Описание
Поддержка связи файлов для запуска и поиска других приложений	В "эпоху" Win 3.x в программе File Explorer было предусмотрено диалоговое окно Associate (Связать), которое давало возможность пользователю связать расширение имени файла с определенным приложением. С появлением Windows 95 программа Windows Explorer "обзавелась" более детализированным диалоговым окном Associate, которое стало называться Open With (Открыть с помощью). Системный реестр обеспечивает автоматическую связь расширений имен файлов и приложений. В окне программы Windows Explorer двойной щелчок на имени любого файла (при наличии его связи с конкретным приложением) вызовет загрузку этого приложения с последующим открытием выбранного файла. За использование связи файла с загружаемым приложением "отвечают" такие функции Shell API, как `FindExecute()` и `ShellExecute()`. Функция `FindExecutable()` используется для считывания имени и дескриптора, чтобы можно было связать приложение с заданным файлом. Функции `ShellExecute` и `ShellExecuteEx` используются для открытия и печати заданного файла. Приложение, необходимое для открытия файла, запускается на основе зафиксированной связи файлов
Выделение пиктограмм	Приложения, библиотеки динамической компоновки и файлы пиктограмм обычно представляются одной или несколькими пиктограммами. Дескриптор пиктограммы можно легко получить с помощью функции Shell API `ExtractIcon`

В текущей версии Windows насчитывается 120 функций оболочки, многие из которых недокументированны. Для просмотра полного списка доступных Win32 API-функций оболочки используйте описанную выше программу Borland `impdef`, вызываемую из командной строки. Некоторые из недокументированных и малоизвестных функций оболочки описаны ниже в этой главе.

Межнациональные средства

Интерфейс Win32 API обеспечивает поддержку разработки на языках, отличных от английского, и распространение на международных рынках. Например, другие языки и специальные символы лучше всего представляются с помощью поддерживаемого в Windows 16-разрядного набора символов Unicode. Функции поддержки иностранных языков (National Language Support — NLS) помогают настроить любое приложение для рынка нужной языковой ориентации. Различные межнациональные особенности интерфейса API описаны в табл. 24.8.

Таблица 24.8. Функции поддержки иностранных языков

Подкатегория	Описание
Символы, определяемые конечным пользователем (End-User-Defined Characters — EUDC)	Обеспечивает поддержку нестандартных символов, которые недоступны в стандартном экране или печатных шрифтах. EUDC включает синтаксис языков Дальнего Востока (например, японского и китайского)
Input Method Editor (IME)	Предоставляет коллекцию функций, сообщений и структур, используемых для поддержки уникода (Unicode) и наборов двубайтовых символов. Работа IME обеспечивается библиотекой `imm32.dll`.
Поддержка иностранных языков (National Language Support)	Обеспечивает поддержку адаптации и перевода приложений в различные среды с языковыми и геополитическими особенностями

Окончание табл. 24.8

Подкатегория	Описание
Уникод и наборы символов (Unicode and Character Sets)	Обеспечивает поддержку шифрования символов, включающих стандарт Unicode. Шифрование символов Unicode позволяет приложениям поддерживать многоязычную обработку текстов

Сетевые службы

Сетевые службы позволяют осуществлять связь по сети между приложениями, работающими на разных компьютерах. Для создания и управления подключениями к таким общим ресурсам, как папки сетевых компьютеров и сетевые принтеры, используются сетевые функции. Сетевые интерфейсы Win32 (Windows Networking, Windows Sockets, NetBIOS, RAS, SNMP и Network DDE) описаны в табл. 24.9.

Таблица 24.9. Сетевые интерфейсы Win32

Сетевой интерфейс	Описание
Организация сетевых возможностей в Windows (Windows Networking — WNet)	Предоставляет набор независимых от сети функций для реализации сетевых возможностей в приложении. Обеспечивается библиотекой сетевого интерфейса Win32 Network Interface DLL (mpr.dll). Имена всех сетевых функций начинаются с приставки `WNet`
Переносимые функции менеджера локальной сети (Ported LAN Manager Functions)	Обеспечивает работу сетевой операционной системы и первоначально был разработан для OS/2-ориентированных серверов. В настоящее время не считается частью Win32 API. Однако множество его функций доступно в Windows благодаря библиотеке `netapi32.dll`
Интерфейс с сетевой базовой системой ввода/вывода (NetBIOS Interface)	Используется для связи по сети с приложениями, работающими на других компьютерах. Работа сетевой базовой системы ввода/вывода (Network Basic Input/Output System — NetBIOS) обеспечивается библиотекой `netapi32.dll`
Динамический обмен данными по сети (Network Dynamic Data Exchange — Network DDE)	Позволяет средствам DDE работать в сетевой среде. Работа обеспечивается интерфейсом WFW DDE Share Interface (`nddeapi.dll`). Все сетевые DDE-функции имеют приставку `NDde`
Служба удаленного доступа (Remote Access Service — RAS)	Позволяет пользователям на удаленных компьютерах подключаться непосредственно к компьютерной сети, получая доступ к одному или нескольким RAS-серверам. Поддерживает коммутируемое соединение (dial-up networking), например с помощью модема. Работа RAS обеспечивается библиотекой `rasapi32.dll`. Все RAS-функции имеют приставки `RAS`
Простой протокол управления сетевыми ресурсами (Simple Network Management Protocol — SNMP)	Используется для обмена информацией, связанной с управлением работой сетевых ресурсов, через стандартный протокол Internet SNMP. За дополнительной информацией обращайтесь к справочной системе Windows
Программные сокеты Windows (Windows Sockets — WinSock)	Используется для обмена данными посредством TCP и UDP. WinSock опирается на UNIX-парадигму программных сокетов Berkeley Software Distribution (BSD). Работа Windows Socket обеспечивается библиотекой `wsock32.dll` (WinSock версия 1.1) или библиотекой `ws2_32.dll` (WinSock версия 2.0)

Структура и функционирование программ в среде Windows

Описанные выше категории функций Win32 API призваны дать общее представление о композиции и возможностях API. Однако прежде чем перейти к реальным примерам, следует ознакомиться с основными правилами сосуществования операционной среды Windows и Win32-приложения. В этом разделе рассматривается базовая структура и функционирование Windows-программы.

Функция `WinMain()`

Функция `WinMain()`, относящаяся к категории управления окнами (Windows Management), — самая важная Win32-функция любого приложения. Тот, кто уже достаточно долго знаком с C++Builder, вероятно, заметил, что функция `WinMain()` расположена в C++-файле, известном как исходный файл проекта (Project Source File). Функция `WinMain()` является входной точкой для всех Windows-приложений; ее присутствие просто необходимо. Эта Win32 API-функция содержится в любой вашей программе и вызывается при запуске приложения. Функция `WinMain()` заменяет стандартное ANSI-объявление `main()`, необходимое для консольных C++-приложений. Borland C++Builder автоматически генерирует функцию `WinMain()`, и (в большинстве случаев) разработчикам не нужно беспокоиться о композиции этой функции, поскольку C++Builder делает эту работу за вас. Однако в некоторых ситуациях полезно вмешаться и внести определенные дополнения в блок функции `WinMain()`. Некоторые из таких ситуаций будут рассмотрены ниже.

Получить представление о композиции функции `WinMain()` — значит, сделать первый шаг к тому, чтобы узнать, как работает Windows. Следующее объявление используется всеми Windows-приложениями, независимо от среды разработки, в которой оно было создано.

```
WINAPI WinMain(HINSTANCE, HINSTANCE, LPSTR, int)
```

Первый параметр идентифицирует дескриптор модуля для вызванного приложения. Дескриптор приложения будет получен при обращении к Windows-функции `GetModuleHandle()`. Дескриптор — очень важный элемент для Win32 API. Ниже мы рассмотрим различные типы дескрипторов, которые использует Windows для эффективного управления приложениями и процессами.

Второй параметр остался в наследство от Win16 API и продолжает существовать в Win32 API в основном ради переносимости 16-разрядных приложений в среду Win32. В Win16 этот параметр используется для идентификации предыдущего экземпляра приложения, если таковой существовал, а в Win32 второй параметр всегда равен значению `NULL`. Поскольку каждый процесс выполняется в уникальном защищенном адресном пространстве, в Win32 каждая программа является независимым процессом, и поэтому не существует второго экземпляра, который мог бы встретиться в том же адресном пространстве. Но это отнюдь не защищает приложение от многократных запусков. При многократных запусках приложения в среде Win32 создается несколько независимых процессов. Чтобы определить, вызвана ли уже другая копия приложения, в функции `WinMain()` рекомендуется создать объект мьютекса, используя функцию `CreateMutex()`, о которой речь пойдет ниже.

С помощью третьего параметра функции `WinMain()` передается указатель на командную строку, которую можно использовать для задания аргументов приложения, используемых во время его выполнения. Существует несколько способов считывания коллекции параметров, заданных в командной строке. Один из них — использовать Win32-функцию `GetCommandLine()`, а

затем "анатомировать" возвращаемую строку, чтобы определить количество параметров и их значения. В C++Builder предусмотрен более простой метод получения аргументов командной строки с помощью функций Borland `ParamCount()` и `ParamStr()` (не связанных с интерфейсом Win32). Функции `ParamCount()` и `ParamStr()` играют роль параметров `argv` и `argc`, передаваемых стандартной функции `main()`, используемой консольными приложениями.

Наконец, четвертый параметр функции `WinMain()` идентифицирует начальное состояние отображаемого окна. Этот параметр (в соответствии со справочной системой Windows) может иметь одно из значений, перечисленных в табл. 24.10.

Таблица 24.10. Константы-сообщения функции `WinMain()`

Значение	Описание
`SW_HIDE`	Скрывает заданное окно и активизирует другое окно
`SW_MINIMIZE`	Сворачивает заданное окно и активизирует окно верхнего уровня в списке системы
`SW_RESTORE`	Активизирует и отображает окно. Если окно свернуто или развернуто, Windows восстанавливает его исходный размер и позицию (аналогично константе `SW_SHOWNORMAL`)
`SW_SHOW`	Активизирует и отображает окно, используя его текущий размер и позицию
`SW_SHOWMAXIMIZED`	Активизирует и отображает окно в развернутом виде
`SW_SHOWMINIMIZED`	Активизирует и отображает окно в виде пиктограммы
`SW_SHOWMINNOACTIVE`	Отображает окно в виде пиктограммы. Активное окно остается активным
`SW_SHOWNA`	Отображает окно в его текущем состоянии. Активное окно остается активным
`SW_SHOWNOACTIVATE`	Отображает окно, используя "самые свежие" значения размера и позиции. Активное окно остается активным
`SW_SHOWNORMAL`	Активизирует и отображает окно. Если окно свернуто или развернуто, Windows восстанавливает его исходный размер и позицию (аналогично константе `SW_RESTORE`)

Дескрипторы окон

При рассмотрении функции `WinMain()` было упомянуто понятие дескриптора. *Дескрипторы* — очень важный аспект интерфейса Win32 API, поскольку все оконные ресурсы идентифицируются дескрипторами определенного типа. Дескрипторы присваиваются модулям, процессам, потокам, окнам, меню, растровым изображениям, пиктограммам, курсорам и цветовым областям. Дескриптор всегда представляется 32-разрядным значением без знака. Дескрипторы обеспечивают средства и такой механизм для управления объектами, как окно и дочерний процесс, а также позволяют передавать входные данные другим приложениям посредством сообщений обратного вызова. Дескрипторы используются во многих примерах, описанных в этой главе.

Сообщения Windows

Сообщения Windows обеспечивают механизм взаимодействия, используемый для передачи данных различным объектам, представленным с помощью дескриптора. Существует много типов встроенных сообщений Windows, которые может отправлять как система Windows, так

и различные приложения. Например, когда пользователь щелкает кнопкой мыши, перемещает ее указатель или изменяет размер активного окна, система генерирует для соответствующего приложения сообщение, индицирующее происшедшее действие. Сообщения используются для уведомлении о поступлении данных, системных изменениях или для непосредственной передачи информации от одного приложения другому. Главное в процессе передачи сообщений — знать дескриптор объекта, которому нужно доставить сообщение.

SendMessage()/PostMessage()

Доставка сообщений выполняется различными способами. Два самых популярных из них — отправить из приложения сообщения Windows с помощью функций `SendMessage()` и `PostMessage()`. Функция `SendMessage()` передает конкретное сообщение либо для одного дескриптора окна, либо для всех окон верхнего уровня. Приложение, вызвавшее функцию `SendMessage()`, ожидает, пока посланное им сообщение не будет обработано окном-получателем. Функция `PostMessage()` выполняет аналогичное действие, но не утруждает себя ожиданием обработки переданного сообщения и возвращается немедленно к отправителю.

Как функции `SendMessage()`, так и функции `PostMessage()` требуются четыре элемента: дескриптор окна-получателя, идентификатор сообщения, который описывает цель сообщения, и два 32-разрядных параметра сообщений. Параметры сообщений можно использовать для передачи дескриптору получателя самой информации или адреса. Значение `LRESULT`, возвращаемое функцией `SendMessage()` или `PostMessage()`, определяет результат обработки сообщения. Например, ненулевое значение указывает на успешное выполнение функции `PostMessage()`. Подробнее см. справку Windows.

```
LRESULT SendMessage(
    HWND hWnd,          // дескриптор окна-получателя
    UINT Msg,           // сообщение для отправки
    WPARAM wParam,      // первый параметр сообщения
    LPARAM lParam       // второй параметр сообщения
   );
```

Идентификаторы сообщений

Существует более 200 встроенных констант, именуемых идентификаторами сообщений Windows (Windows Message Identifiers). Идентификаторы сообщений Windows часто используются для реализации обработки прерываний между приложениями и операционной системой. Помимо встроенных сообщений Windows, могут быть определены и нестандартные сообщения Windows. Для определения нестандартного сообщения Windows рекомендуется использовать функцию `RegisterWindowMessage()` (из категории Windows Management), чтобы иметь гарантию ее уникальности в системе.

Большинство встроенных идентификаторов сообщений Windows начинается с приставки `WM_`. Ниже приводятся самые популярные сообщения.

- `WM_PAINT`. Сообщение-команда перерисовать окно.
- `WM_QUIT`. Сообщение-команда убрать приложение из очереди.
- `WM_LBUTTONDBLCLK`. Был сделан двойной щелчок левой кнопкой мыши.
- `WM_LBUTTONDOWN`. Была нажата левая кнопка мыши.
- `WM_LBUTTONUP`. Была отпущена левая кнопка мыши.
- `WM_MBUTTONDBLCLK`. Был сделан двойной щелчок средней кнопкой мыши.
- `WM_MBUTTONDOWN`. Была нажата средняя кнопка мыши.

- WM_MBUTTONUP. Была отпущена средняя кнопка мыши.
- WM_RBUTTONDBLCLK. Был сделан двойной щелчок правой кнопкой мыши.
- WM_RBUTTONDOWN. Была нажата правая кнопка мыши.
- WM_RBUTTONUP. Была отпущена правая кнопка мыши.
- WM_NCHITTEST. Только что произошло событие мыши.
- WM_KEYDOWN. Была нажата несистемная клавиша.
- WM_TIMER. Сообщение о событии таймера.

Ответ на сообщения Windows

Чтобы ответить на сообщение, посланное функцией SendMessage() или PostMessage(), приложению нужно иметь специальную таблицу ответов на события. Существует несколько способов установки таблицы такого типа. Самый популярный способ в среде C++Builder — объявить функции обратного вызова и карту сообщений в рамках защищенной области объявления класса главной формы. Вот пример.

```
protected:     // Объявления пользователя.
// Сообщения обратного вызова.
    void __fastcall Process_wmPaint(TMessage &);
    void __fastcall Process_wmQuit(TMessage &);
    void __fastcall Process_wmXYZ(TMessage &);
    BEGIN_MESSAGE_MAP
       MESSAGE_HANDLER(WM_PAINT,TMessage,Process_wmPaint);
       MESSAGE_HANDLER(WM_QUIT,TMessage,Process_wmQuit);
       MESSAGE_HANDLER(WM_XYZ,TMessage,Process_wmXYZ);
    END_MESSAGE_MAP(TForm);
```

В исходном тексте приложения функция обратного вызова должна выглядеть примерно так.

```
void __fastcall Tform1::Process_wmXYZ(TMessage Msg)
{
    int fromhandle = LOBYTE(LOWORD(Msg.WParam));
    int infoid = Msg.LParam;
    AnsiString StatusText =
      " Application sent message XYZ\ nApplication Handle =
                                " + BIntToStr(fromhandle) + "\ nInfo =
                                                    " + IntToStr(infoid);

    MessageBox(Handle, StatusText, "Received Message Callback", MB_OK);
}
```

Тип Borland TMessage, определенный ниже, предоставляет структуру для сообщения Windows.

```
struct TMessage
{
    Cardinal Msg;
    union
    {
        struct
```

```
        {
            Word WParamLo;
            Word WParamHi;
            Word LParamLo;
            Word LParamHi;
            Word ResultLo;
            Word ResultHi;
        } ;
        struct
        {
            int WParam;
            int LParam;
            int Result;
        } ;

    } ;
} ;
```

Обычно сообщение, переданное с помощью функции `SendMessage()` или `PostMessage()`, будет содержать информацию, заключенную в параметрах `WParam` и `LParam`. Функция обратного вызова должна проанализировать и дешифрировать эти 32-разрядные значения. Важно отметить, что одно из этих 32-разрядных значений может потенциально являться указателем на доступный адрес, если источник вызова размещен внутри того же процесса, что и получатель сообщения (в Win16 адресное пространство было доступно для нескольких процессов, в то время как адресное пространство Win32 обладает большей защитой). При передаче указателя (вместо данных) другим объектам возможен доступ не только к 32-разрядному элементу, но и к данным большего объема.

Реальные примеры использования интерфейса API

Лучший способ понять, как использовать интерфейс Win32 API в процессе C++Builder-разработки, — разобраться в нескольких реальных примерах. При рассмотрении этих примеров ваше внимание будет обращено на ключевые функции, используемые разработчиками для создания и поддержки ошибкоустойчивых приложений. Кроме того, вы познакомитесь с некоторыми недокументированными API-функциями, которые вошли в Windows 2000.

Запуск приложения в программе

Часто возникают ситуации, когда программным путем было бы удобно запускать другие приложения. Существует несколько Win32-функций, позволяющих запускать приложения, а именно: `WinExec()`, `LoadModule()`, `CreateProcess()`, `ShellExecute()` и `ShellExecuteEx()`.

Функция `WinExec()` — старая "любимица" многих Windows-разработчиков, однако Microsoft рекомендует разработчикам воздерживаться от использования функций `WinExec()` и `LoadModule()`, а вместо них использовать функцию `CreateProccess()`, которая в действительности и вызывается в реализациях функций `WinExec()` и `LoadModule()`. Существует мнение, что обе эти функции (которые достались в наследство от Win16) в конце концов будут удалены из будущих спецификаций Win API. С другой стороны, функции `ShellExecute()` и

ShellExecuteEx() реализуют совсем иной механизм, в котором используются 32-разрядные значения системного реестра Windows для определения способа открытия файла. Функции ShellExecute() и ShellExecuteEx() относятся к описанной выше категории функций оболочки (Windows Shell). У функции ShellExecuteEx() более широкий спектр действий по сравнению с функцией ShellExecute(). В этом разделе описаны примеры использования функций CreateProcess() и ShellExecuteExe().

Использование функции CreateProcess() для получения дескрипторов окон

Рекомендованный компанией Microsoft способ запуска приложения в среде Windows заключается в использовании метода CreateProcess(). Несмотря на эту рекомендацию, вы должны знать, что это самый трудный способ, хотя он обладает определенными преимуществами перед другими методами.

Предположим, в приложении нам нужно не только запускать другие приложения, но и управлять ими. Например, когда главное приложение будет свернуто, нужно обеспечить возможность свернуть и все другие. Или же, когда главное приложение будет закрыто, нужно обеспечить возможность закрыть и все другие приложения, запущенные из главного. Для поддержки возможностей управления приложениями важно определить (или найти) дескриптор другого приложения. Знание дескриптора нужного приложения позволит передать ему стандартные либо пользовательские (нестандартные) сообщения Windows. Для облегчения поиска дескриптора приложения можно использовать функцию CreateProcess(). В следующем фрагменте программы показан один из способов выполнения приложения и получения его дескриптора.

```
processid = RunProgramEx(filelocation,parameters);
// Возможно, стоит подождать 100 миллисекунд
if (processid != 0)
{
     winhandle = LookForWindowHandle(Apps[AppCount].processid);
     GetWindowText( Apps[AppCount].winhandle, windowtitle, 80);
     MessageBox(Handle, windowtitle, "Приложение запущено ", MB_OK);
}
```

Показанный в этом примере вызов функции-“матрешки” RunProgramEx() используется специально для “ввода в игру” приложения и получения ID процесса. Внутри ее скрыт вызов функции CreateProcess(). Функция LookForWindowHandle() — это также нестандартная “матрешка”, которая просматривает список активных приложений, “обращая внимание” на значение ID процесса. При обнаружении совпадения становится возможным получение дескриптора приложения. Обе эти функции-“матрешки” инкапсулируют несколько функций Win32 API различных категорий, которые будут подробно описаны в следующих разделах.

Функция Win32 API GetWindowText() используется для получения заголовка окна только что запущенного приложения. Чтобы уведомить пользователя о запуске приложения, вызывается функция Windows MessageBox(), которая выводит на экран этот заголовок. А теперь стоит рассмотреть нестандартную функцию RunProgramEx(), приведенную в листинге 24.1.

Листинг 24.1. Функция RunProgramEx()

```
unsigned long RunProgramEx (char* file, char* parameters)
{
```

```
    STARTUPINFO  StartupInfo;
    ZeroMemory( &StartupInfo, sizeof(STARTUPINFO));
    PROCESS_INFORMATION ProcessInfo;
    StartupInfo.cb = sizeof(STARTUPINFO);
    if(CreateProcess(file,
        parameters,
        NULL,
        NULL,
        TRUE,
        NORMAL_PRIORITY_CLASS,
        NULL,
        NULL,
        &StartupInfo,
        &ProcessInfo))
    {
        // Мы должны закрыть дескрипторы, возвращенные в
        // ProcessInfo. Мы можем закрыть дескриптор в любое
        // время, так почему бы не сделать это сейчас.
        CloseHandle(ProcessInfo.hProcess);
        CloseHandle(ProcessInfo.hThread);
        return (unsigned long)ProcessInfo.dwProcessId;
    }
    return 0;
}
```

Функция `RunProgramEx()` инкапсулирует несколько Win32 API-функций, включая `ZeroMemory()`, `CreateProcess()` и `CloseHandle()`. Самой важной из них является функция `CreateProcess()`. Полностью соответствуя своему имени, она создает новый процесс приложения и его поток выполнения, запустив заданный выполняемый файл. Имя файла и аргументы времени выполнения передаются как параметры функции `RunProgramEx()`. `StartUpInfo` — это передаваемая функции `CreateProcess()` переменная, которая определяет, каким способом появится главное окно для нового приложения. Переменная `StartUpInfo` инициализируется с помощью Win32-функции `ZeroMemory()`, которая просто заполняет структуру нулями. Переменная `StartUpInfo` определяется структурой `STARTUPINFO`, как описано в справке Win32 API.

Функция `CreateProcess()` заполняет переменную `ProcessInfo` информацией о только что созданном процессе и его основном потоке. Переменная `ProcessInfo` определяется структурой `PROCESS_INFORMATION`, представленной ниже.

```
typedef struct _PROCESS_INFORMATION
{
    HANDLE hProcess;
    HANDLE hThread;
    DWORD dwProcessId;
    DWORD dwThreadId;
}  PROCESS_INFORMATION;
```

Идентификатор (ID) процесса, возвращаемый нестандартной функцией `RunProgramEx()`, на самом деле выделяется из поля `dwProcessId` этой структуры. А затем с помощью значения ID и посредством функции `LookForWindowHandle()` и определяется Windows-дескриптор запущенного приложения.

```
HWND LookForWindowHandle(unsigned long processid)
{
        if (!EnumWindows((WNDENUMPROC)GetWinHandle,processid))
            return swProcess;
        else
            return 0;
}
```

Функция LookForWindowHandle() — это пользовательская функция, вызывающая функцию Win32 EnumWindows(), которая опрашивает все окна верхнего уровня, используя при этом пользовательскую функцию обратного вызова GetWinHandle(), указанную в первом параметре вызова функции EnumWindows() (см. листинг 24.2).

Листинг 24.2. Обход окон с помощью функции GetWinHandle()

```
HWND swProcess;

BOOL CALLBACK GetWinHandle(HWND hwnd, unsigned long hproc)
{
    char windowtitle[80];
    char classname[80];
    GetWindowLong(hwnd,GWL_HINSTANCE);
    unsigned long dwProcessId;    // Адрес переменной для
                                        // идентификатора процесса.
    GetWindowThreadProcessId(
        hwnd,
        &dwProcessId);     // Адрес переменной для ID процесса.
    if (dwProcessId != hproc)
        return true;
    GetWindowText( hwnd, windowtitle, 80);
    if (windowtitle[0] == NULL)
        return true;

    GetClassName(hwnd, classname, 80);
    int ptr = strcmp(classname, "TApplication");
    if (ptr == 0)
        return true;

    swProcess=hwnd;

    return FALSE; //Прекращение обхода: дескриптор уже получен!
}
```

Нестандартная функция GetWinHandle() считывает дескриптор окна на основе информации о процессе. Когда функция EnumWindows() возвращает ненулевое значение, дескриптор окна возвращается функцией LookForWindowHandle().

Использование функции ShellExecuteEx()

Есть более простой способ запустить приложение. Для этого можно воспользоваться функцией ShellExecuteEx() из библиотеки Shell, как показано в листинге 24.3. Хотя ее легче использовать, чем функцию CreateProcess(), но она не позволяет получить дескриптор за-

пускаемого приложения. Следующая пользовательская функция-"матрешка" показана с целью демонстрации возможности ее использования.

Листинг 24.3. Запуск приложения с помощью функции `ShellExecuteEx()`

```
int  RunProgram (char* file, char* parameters)
{
  // Выбираем программу и характер ее выполнения.
  SHELLEXECUTEINFO execinfo ;
  memset (&execinfo, 0, sizeof (execinfo)) ;
  execinfo.cbSize = sizeof (execinfo) ;
  execinfo.lpVerb = "open" ;
  execinfo.lpFile = file;
  execinfo.lpParameters = parameters;
  execinfo.fMask = SEE_MASK_NOCLOSEPROCESS ;
  execinfo.nShow = SW_SHOWDEFAULT ;

  // Выполняем программу.
  if (! ShellExecuteEx (&execinfo))
  {
    char data[100];
    sprintf(data, "Could not run program '%s'", file);
    MessageBox(NULL, data, "Operation Error!", MB_OK);
    return -1;
  }
  return 0;
}
```

Структура `execinfo` типа `SHELLEXECUTEINFO` передается в качестве параметра функции `ShellExecuteEx()`. Двумя ключевыми атрибутами структуры `execinfo` являются флаг `open` и указатель `file`. Аргументы командной строки также можно передать как параметры. Функции `ShellExecute()` и `ShellExecuteEx()` для открытия файлов и запуска приложений используют зафиксированную в системе связь файлов (на основе расширения файла). Ниже мы рассмотрим другие функции библиотеки Shell.

Базовые операции ввода-вывода файлов

Ввод-вывод файлов (чтение и запись данных на диск) является жизненно важным аспектом использования вычислительной техники. Такие функции Win32, как `ReadFile()`, `WriteFile()` и `CreateFile()` используются для передачи данных на диск и обратно. Примеры с использованием этих функций мы и рассмотрим в этом разделе.

`CreateFile()`

Функция `CreateFile()` используется для создания объектов и возврата их дескрипторов вызывающей функции. Функция `CreateFile()` поддерживает не только файлы, но и консоли, каналы и другие элементы связи между процессами (Interprocess Communications — IPC). Однако в этом разделе мы хотим продемонстрировать ее использование для работы с файлами. Получив представление о назначении функции `CreateFile()`, вы непременно захотите применить ее для поддержки IPC и других возможностей.

Функция `CreateFile()` имеет следующий формат.

```
HANDLE CreateFile(
    LPCTSTR lpFileName,       // Указатель на имя файла.
    DWORD dwDesiredAccess,    // Режим доступа (чтение-запись).
    DWORD dwShareMode,        // Режим совместного использования.
    LPSECURITY_ATTRIBUTES lpSecurityAttributes,  // Атрибуты
                                                 // безопасности.
    DWORD dwCreation,             // Способ создания.
    DWORD dwFlagsAndAttributes,   // Атрибуты файла.
    HANDLE hTemplateFile          // Дескриптор для копируемого
                                  // файла с атрибутами.
  );
```

lpFileName — указатель на строку с завершающим нулевым символом, которая должна содержать имя объекта, подлежащего открытию или созданию. Максимальная длина этой строки определяется значением константы MAX_PATH (255 символов).

dwDesiredAccess — параметр, который определяет тип доступа к объекту. Вы можете выполнять чтение, запись или формировать запросы к объекту. Возможные типы доступа представлены в табл. 24.11.

Таблица 24.11. Типы доступа к объекту

Значение	Описание
0	Задает доступ к объекту для запроса информации об устройстве. Можно запросить атрибуты устройства, не получая доступа к реальному устройству
GENERIC_READ	Позволяет считывать данные и управлять файловым указателем. Для доступа в режиме чтения и записи можно использовать конъюнкцию с константой GENERIC_WRITE
GENERIC_WRITE	Позволяет записывать данные и управлять файловым указателем. Для доступа в режиме чтения и записи можно использовать конъюнкцию с константой GENERIC_READ

Параметр dwShareMode идентифицирует характер совместного использования объекта. Его значения перечислены в табл. 24.12.

Таблица 24.12. Типы совместного использования объекта

Значение	Описание
0	Не допускает совместного использования файла
FILE_SHARE_READ	Разрешены другие операции чтения. Это значение задается при открытии почтового канала (mailslot) на стороне клиента
FILE_SHARE_WRITE	Разрешены другие операции по открытию файла для доступа к записи

lpSecurityAttributes — указатель на структуру SECURITY_ATTRIBUTES, которая определяет безопасность для заданной папки. Этот параметр можно использовать только в среде Windows NT. (Windows NT поддерживает безопасность в файловой системе NTFS). Этот параметр не окажет никакого влияния на систему, которая его не поддерживает.

Параметр dwCreation означает действие, которое будет предпринято в случае существования файла или его отсутствия. Список возможных значений приведен в табл. 24.13.

Таблица 24.13. Значения параметра dwCreation

Значение	Описание
CREATE_NEW	Создает новый файл. Эта функция не срабатывает, если указанный файл уже существует
CREATE_ALWAYS	Создает новый файл и перезаписывает предыдущий, если таковой уже существует
OPEN_EXISTING	Открывает существующий заданный файл. Эта функция не срабатывает, если указанный файл не существует
OPEN_ALWAYS	Открывает файл, если таковой уже существует. В противном случае эта функция создает файл, как если бы параметр dwCreation был равен значению CREATE_NEW
TRUNCATE_EXISTING	Открывает файл. После того как файл открыт, он усекается до нуля байт. Вызывающий процесс в этом случае должен открывать файл, установив режим доступа равным значению GENERIC_WRITE. Если файл не существует, эта функция не срабатывает

Параметр dwFlagsandAttributes задает атрибуты и флаги для файла. Можно использовать любую комбинацию атрибутов, приведенных в табл. 24.14, но все другие атрибуты переопределяют атрибут FILE_ATTRIBUTE_NORMAL.

Таблица 24.14. Флаги и атрибуты файлов

Значение	Описание
FILE_ATTRIBUTE_ARCHIVE	Файл следует заархивировать. Приложения используют этот атрибут, чтобы отметить файлы для резервирования или удаления
FILE_ATTRIBUTE_NORMAL	Для файла не установлены никакие другие атрибуты. Этот атрибут действителен при условии что используется только он один
FILE_ATTRIBUTE_HIDDEN	Файл скрыт
FILE_ATTRIBUTE_READONLY	Файл предназначен только для чтения
FILE_ATTRIBUTE_SYSTEM	Этот файл системный. Он используется исключительно операционной системой или некоторой ее частью
FILE_ATTRIBUTE_TEMPORARY	Файл предназначен для временного хранения
FILE_ATTRIBUTE_ATOMIC_WRITE	Зарезервирован. Не используется
FILE_ATTRIBUTE_XACTION_WRITE	Зарезервирован. Не используется
FILE_ATTRIBUTE_OFFLINE	Этот файл в данный момент автономен (offline). Физически он был перемещен
FILE_ATTRIBUTE_COMPRESSED	Файл или папка в сжатом состоянии. Для файла это значит, что данные упакованы, а для папки — что сжатие будет происходить по умолчанию для вновь создаваемых файлов или подпапок
FILE_FLAG_WRITE_THROUGH	Указывает операционной системе на запись через любой промежуточный кэш и прямо на диск

Окончание табл. 24.14

Значение	Описание
FILE_FLAG_OVERLAPPED	Указывает операционной системе на инициализацию объекта, поэтому операции, требующие большого времени выполнения (например ReadFile(), WriteFile(), ConnectNamedPipe() и TransactNamedPipe()) возвращают значение ERROR_IO_PENDING. По завершении операции событие устанавливается в "сигнальное" состояние. При использовании этого флага необходимо определить структуру OVERLAPPED, т.е., когда задается флаг FILE_FLAG_OVERLAPPED, приложение должно совмещать операции чтения и записи.
	При использовании этого флага операционная система не поддерживает файловый указатель. Позицию в файле нужно передавать как часть параметра lpOverlapped (указывающую на структуру OVERLAPPED) для функций ReadFile() и WriteFile()
FILE_FLAG_NO_BUFFERING	Предписывает операционной системе открыть файл без промежуточной буферизации или кэширования
FILE_FLAG_RANDOM_ACCESS	Означает произвольное использование файла
FILE_FLAG_SEQUENTIAL_SCAN	Использование файла с последовательным чтением/записью от начала к концу
FILE_FLAG_DELETE_ON_CLOSE	Операционная система удаляет файл немедленно после закрытия *всех* его дескрипторов, а не только дескриптора, для которого был задан флаг FILE_FLAG_DELETE_ON_CLOSE
FILE_FLAG_BACKUP_SEMANTICS	Означает, что файл открывается или создается для операции резервирования или восстановления. Этот флаг можно использовать только под управлением Windows NT
FILE_FLAG_POSIX_SEMANTICS	Означает, что к файлу возможен доступ в соответствии с правилами POSIX (Portable Operating System Interface for computer environments — интерфейс переносимой операционной системы)

Если функция CreateFile() открывает на стороне клиента именованный канал, параметр dwFlagsAndAttributes может также содержать информацию об уровне безопасности обслуживания (Security Quality of Service — SQOS). Когда вызывающее приложение задает флаг SECURITY_SQOS_PRESENT, параметр dwFlagsAndAttributes может содержать одно или несколько значений, перечисленных в табл. 24.15.

Таблица 24.15. Доступные флаги файлов для установки уровня безопасности обслуживания (SQOS)

Значение	Описание
SECURITY_ANONYMOUS	Обслуживает клиента на неперсонифицированном уровне Anonymous
SECURITY_IDENTIFICATION	Обслуживает клиента на неперсонифицированном уровне Identification
SECURITY_IMPERSONATION	Обслуживает клиента на уровне Impersonation

Окончание табл. 24.15

Значение	Описание
SECURITY_DELEGATION	Обслуживает клиента на неперсонифицированном уровне Delegation
SECURITY_CONTEXT_TRACKING	Указывает, что режим отслеживания безопасности (Security Tracking Mode) является динамическим. Если этот флаг не задан, режим отслеживания безопасности является статическим
SECURITY_EFFECTIVE_ONLY	Указывает, что только разрешенные аспекты контекста безопасности клиента доступны для сервера. Если этот флаг не задан, все аспекты контекста безопасности клиента доступны. Этот флаг позволяет клиенту ограничить группы и привилегии, которые сервер может использовать при обслуживании клиента

hTemplateFile — это дескриптор с доступом GENERIC_READ к файлу шаблона. Шаблонный файл поддерживает расширенную информацию об атрибутах для создаваемого файла. Этот параметр не поддерживается в Windows 95/98, поскольку эти операционные системы не обеспечивают безопасность файловой системы. Этот параметр можно использовать только в Windows NT.

Использовать функцию CreateFile() несложно. В следующем фрагменте программы показано, как использовать функцию CreateFile(), которая возвращает дескриптор файла.

```
HANDLE afile;
Afile = CreateFile("FILE.TXT" GENERIC_READ | GENERIC_WRITE, 0,
                   NULL, BOPEN_EXISTING,
                   FILE_ATTRIBUTE_NORMAL, NULL);
```

В этом маленьком примере файл FILE.TXT открывается в режиме доступа как для чтения, так и для записи, причем этот файл не подлежит совместному использованию, он должен существовать (в противном случае функция не срабатывает), и в нем не предусмотрены никакие атрибуты.

ReadFile()

Функция ReadFile() читает данные из файла с начальной позиции файлового указателя. Файловый указатель перемещается в новую позицию в соответствии с количеством прочитанных байт. Если файл был создан с флагом совмещенных атрибутов (FILE_FLAG_OVERLAPPED), файловый указатель настраивается приложением с помощью функции SetFilePointer(). Подробнее по использованию флага FILE_FLAG_OVERLAPPED можно прочитать в разделе, посвященном описанию параметра lpOverLapped.

Функция ReadFile() принимает следующие параметры.

```
BOOL ReadFile(
    HANDLE hFile,                  // Дескриптор читаемого файла.
    LPVOID lpBuffer,               // Адрес буфера, который принимает данные.
    DWORD nNumberOfBytesToRead,    // Количество байт, которые нужно прочитать.
    LPDWORD lpNumberOfBytesRead,   // Адрес количества прочитанных байт.
    LPOVERLAPPED lpOverlapped      // Адрес структуры для данных.
   );
```

hFile — дескриптор читаемого файла.

lpBuffer — указатель на буфер, принимающий данные.

nNumberOfBytesToRead — количество байт, читаемых из файла.

lpOverlapped — указатель на структуру OVERLAPPED. Здесь мы не используем параметры для структуры OVERLAPPED. Если дескриптор для файла не создается с использованием флага FILE_FLAG_OVERLAPPED и при этом указатель lpOverlapped не равен значению NULL, операция записи начинается со смещения, заданного в структуре OVERLAPPED, а функция WriteFile() может вернуться до завершения записи. В этом случае функция WriteFile() возвращает значение FALSE, а функция GetLastError() — значение ERROR_IO_PENDING. Это позволяет вызвавшему ее процессу продолжить обработку до полного завершения записи. На это время событие, заданное в структуре OVERLAPPED, устанавливается в "сигнальное" состояние. Ниже приведен формат структуры OVERLAPPED. Более подробно об этом можно прочитать в справке Win32 API, входящей в состав справочной системы C++Builder.

Если файл, заданный дескриптором hFile, был открыт без использования флага FILE_FLAG_OVERLAPPED, а значение указателя lpOverlapped равно NULL, запись начинается с текущей позиции файла, и функция WriteFile() не возвратится в вызвавший ее процесс до тех пор, пока операция не будет полностью завершена.

Если файл, заданный дескриптором hFile, не был открыт с использованием флага FILE_FLAG_OVERLAPPED, а значение указателя lpOverlapped не равно NULL, запись начинается со смещения, заданного в структуре OVERLAPPED, а функция WriteFile() не возвратится в вызвавший ее процесс до тех пор, пока операция не будет полностью завершена.

Структура OVERLAPPED содержит информацию, используемую в операциях асинхронного ввода и вывода (I/O).

```
typedef struct _OVERLAPPED {
    DWORD  Internal;
    DWORD  InternalHigh;
    DWORD  Offset;
    DWORD  OffsetHigh;
    HANDLE hEvent;
}  OVERLAPPED;
```

Поле Internal резервируется для использования операционной системой. Этот член, который задает состояние, определяемое системой, действителен, когда функция GetOverlappedResult() возвращается без установки расширенной информации об ошибках равной значению ERROR_IO_PENDING.

Поле InternalHigh резервируется для использования операционной системой. Этот член, который задает длину переданных данных, действителен, когда функция GetOverlappedResult() возвращает значение TRUE.

Поле Offset задает файловую позицию, с которой нужно начинать передачу. Файловая позиция представляет собой байтовое смещение от начала файла. Вызывающий процесс устанавливает этот член до вызова функции ReadFile() или WriteFile(). Этот член игнорируется при чтении из именованных каналов и коммуникационных устройств или записи в них.

Поле OffsetHigh задает старшее слово байтового смещения, с которого нужно начинать передачу. Этот член игнорируется при чтении из именованных каналов и коммуникационных устройств или записи в них.

Поле HEvent идентифицирует набор событий для сигнализации о завершении передачи данных. Вызывающий процесс устанавливает этот член до вызова любой из функций: ReadFile(), WriteFile(), ConnectNamedPipe() или TransactNamedPipe().

При успешном выполнении функция ReadFile() возвращает значение TRUE; в противном случае — значение FALSE.

WriteFile()

Функция WriteFile() записывает данные на диск, начиная с позиции файлового указателя. По завершении записи файловый указатель перемещается в соответствии с количеством записанных байт. Подобно функции ReadFile(), если файл открыт с использованием флага FILE_FLAG_OVERLAPPED, файловый указатель должен настраиваться приложением. Здесь мы не будем использовать этот флаг.

Функция WriteFile() принимает следующие параметры.

```
BOOL WriteFile(
    HANDLE hFile,                    // Дескриптор файла, в который
                                     // записываются данные.
    LPCVOID lpBuffer,                // Указатель на данные, записываемые в файл.
    DWORD nNumberOfBytesToWrite,     // Количество записываемых байт.
    LPDWORD lpNumberOfBytesWritten,  // Указатель на количество записанных байт.
    LPOVERLAPPED lpOverlapped        // Указатель на структуру, необходимую для
                                     // совмещенного ввода/вывода.
  );
```

hFile — дескриптор файла.

lpBuffer — буфер, содержащий данные, отправляемые на диск.

nNumberOfBytesToWrite — количество байт, записываемых на диск.

lpNumberOfBytesWritten — количество байт, записанных на диск.

lpOverlapped — указатель на структуру, необходимую в случае, если дескриптор для файла был открыт с флагом FILE_FLAG_OVERLAPPED.

При успешном выполнении функция WriteFile() возвращает значение TRUE, в противном случае — значение FALSE.

Теперь можно рассмотреть пример, демонстрирующий чтение и запись данных из файла. Начнем с загрузки C++Builder и создания нового приложения.

Поместите компоненты в форму, как показано в табл. 24.16.

Таблица 24.16. Список компонентов

Компонент	Свойство
Form1	Left = 235 Top = 184
Label1	Left = 9 Top = 9 Width = 124 Caption = Введите имя создаваемого файла:
Label2	Left = 8 Top = 130 Width = 262 Caption = При щелчке на кнопке CREATE содержимое буфера будет записано и отображено в нижнем заголовке. При щелчке на кнопке READ данные будут прочтены и отображены. WordWrap = True
Label5	Name = BufferLabel1 Left = 8 Top = 205 Caption = Buffer

Окончание табл. 24.16

Компонент	Свойство
Label3	Left = 6 Top = 186 Caption = Данные в буфере---
Label4	Left = 10 Top = 56 Caption = Введите сюда какой-нибудь текст для тестирования буфера.
Edit1	Left = 7 Top = 25 Width = 265 Height = 21 Text = c:\ test.txt
Button1	Left = 13 Top = 98 Width = 75 Height = 25 Caption = &Create
Button2	Left = 200 Top = 98 Width = 75 Height = 25 Caption = &Read
Edit2	Left = 8 Top = 72 MaxLength = 255 Text = Это пример текста, который будет записан в файл.

Сохраните проект под именем `readwrite`. Сохраните модуль как файл `mainform.cpp`. Вставьте текст программы из листинга 24.4.

Листинг 24.4. Примеры использования функций `ReadFile()`, `WriteFile()` и `CreateFile()`

```
//---------------------------------------------------------------
#include <vcl\ vcl.h>
#pragma hdrstop
#include "mainform.h"
//---------------------------------------------------------------
#pragma resource "*.dfm"
TForm1 *Form1;
   HANDLE hSrc;      // Создаем дескриптор файла для
                     // выполнения чтения и записи.
//---------------------------------------------------------------
__fastcall TForm1::TForm1(TComponent* Owner): TForm(Owner)
{
}
//---------------------------------------------------------------
```

```
void __fastcall TForm1::Button1Click(TObject *Sender)
{
   char buffer[255];       // Буфер для хранения данных.
   DWORD  bytes_written;   // Количество записываемых данных.

    /* Подготовка к открытию файла с доступом для записи. Флаг
       CREATE_ALWAYS означает, что файл будет создан всегда, если
       даже для этого потребуется перезаписать старый файл.
       Мы откроем файл с произвольным доступом.
    */
   hSrc = CreateFile(Edit1->Text.c_str(), GENERIC_WRITE, 0, NULL,
           CREATE_ALWAYS, FILE_FLAG_RANDOM_ACCESS, 0);

  // Если дескриптор файла окажется неверным, генерируем
  // сообщение об ошибке.
  if(hSrc == INVALID_HANDLE_VALUE)
    {
     Application->MessageBoxA("Ошибка при открытии файла", NULL,
                                                    NULL); return;
    }

  // Копируем содержимое строки редактирования EditBox в буфер.
  strcpy(buffer, Edit2->Text.c_str());

  // Теперь выполняем запись....
  WriteFile(hSrc, buffer, sizeof(buffer), &bytes_written, NULL);
  // Закрываем файл после записи...
  CloseHandle(hSrc);

  // Помещаем буфер в заголовок.
  BufferLabel->Caption = buffer;
}
//---------------------------------------------------------------
void __fastcall TForm1::Button2Click(TObject *Sender)
{
   BufferLabel->Caption = "";

   // Объявляем буфер для приема данных.
   char Buffer[255];
   DWORD dwRead;

  // Открываем файл с доступом для чтения, без атрибутов,
  // и если этот файл существует...
   hSrc = CreateFile(Edit1->Text.c_str(), GENERIC_READ, 0, NULL,
           OPEN_EXISTING, FILE_ATTRIBUTE_NORMAL, 0);

  if(hSrc == INVALID_HANDLE_VALUE)
   {
    Application->MessageBoxA("Ошибка при открытии файла", NULL, NULL); return;
   }
```

```
  if(ReadFile(hSrc , Buffer, sizeof(Buffer), &dwRead, NULL) )
    {
      BufferLabel->Caption =Buffer;
      ShowMessage("С чтением все ОК");
    }
     else
     {  ShowMessage("Нет, не читается");  }
    CloseHandle (hSrc);
}
```

В листинге 24.4, чтобы создать дескриптор для файла, не забудьте ввести строку `HANDLE hSrc;`. Обратите внимание, что в программе предусмотрены события, связанные с кнопками, после щелчка на которых и будет выполнена вся работа. События можно создать автоматически, дважды щелкнув на соответствующей кнопке. C++Builder автоматически генерирует код в заголовочном файле и главном файле исходного кода.

Теперь скомпилируйте и выполните эту программу. Как видите, записывать в файл и читать из него данные — проще пареной репы.

В интерфейсе Win32 API существуют и другие функции, которые можно использовать для работы с файлами, например `MoveFile()`, `DeleteFile()`, и несколько новых `SHx`-функций. `SHx`-функции — это более новые функции системной оболочки (Shell), предусмотренные в Windows 95 и Windows NT 4. Однако сначала (из уважения к "возрасту") исследуем некоторые старые функции управления файлами, а затем перейдем к более новым Shell-функциям, которые можно использовать для выполнения аналогичных задач.

MoveFile()

Функция `MoveFile()` используется для перемещения существующего файла или папки в другое место запоминающего устройства подобного типа. Однако функция `MoveFile()` окажется бессильной, если тип приемника отличается от типа источника. Дело в том, что папки на одинаковых запоминающих устройствах на самом деле не переносятся, а переименовываются.

Функция `MoveFile()` принимает следующие параметры.

```
Bool MoveFile(LPCTSTR lpszExisting  , LPCTSTR lpszNew);
```

`lpszExisting` — указатель на строку с завершающим нулевым символом, которая содержит имя существующего файла или папки.

`lpszNew` — указатель на строку с завершающим нулевым символом, которая содержит имя нового файла или папки. Имя приемника не должно существовать.

При успешном выполнении функция возвращает значение `TRUE`, в противном случае — значение `FALSE`.

```
MoveFile("C:\\Winters\\Lauren\\Pics",
                                    "c:\\Sexton\\Family\\Pics\\Backup");
```

DeleteFile()

Функция `DeleteFile()` работает аналогично стандартной ANSI C/C++-функции `unlink()`. Она удаляет файл с устройства. Если она не может удалить существующий файл или этот файл в данный момент открыт либо не существует, функция возвратит значение `FALSE`.

Функция `DeleteFile()` принимает следующие параметры.

```
bool DeleteFile(LPCSTR lpszFileName)
```

lpszFileName — указатель на строку с завершающим нулевым символом, которая содержит имя файла.

```
DeleteFile("c:\\Collin\\TextFiles\\music.txt");
```

CopyFile()

Функция CopyFile() копирует файлы в другое место запоминающего устройства. Атрибуты копируются вместе с файлом, за исключением атрибутов безопасности.

Функция CopyFile() имеет следующие параметры.

```
bool LPCSTR lpszExistingfile , LPCSTR lpsz NewFile,
                                          BOOL FailIfExisting)
```

lpszExsistingFile — имя существующего файла, подлежащего копированию.

lpszNewFile — указатель на строку с завершающим нулевым символом, которая содержит имя файла приемника.

Параметр failIfExisting означает действия, которые выполняются в случае существования файла приемника. Если этот параметр равен значению TRUE, функция не выполнит свою миссию, т.е. этот файл не будет перезаписан. Если передано значение FALSE, файл будет перезаписан.

```
CopyFile("file.txt" , "file.txt" , true);
```

CreateDirectory()

Функция CreateDirectory() создает папку на запоминающем устройстве. Если операционная система поддерживает безопасность, функция CreateDirectory() применяет дескриптор безопасности к новой папке. В отличие от Windows NT, Windows 9x не поддерживает безопасность.

Функция CreateDirectory() принимает следующие параметры.

```
bool CreateDirectory(LPCSTR lpszPath,
                     LPSECURITYATTRIBUTES lpSecurityAttributes)
```

lpszPath — указатель на строку с завершающим нулевым символом, которая содержит имя новой папки.

lpSecurityAttributes — указатель на структуру, которая определяет атрибуты безопасности для папки. Файловая система должна поддерживать средства безопасности.

```
CreateDirectory("C:\\Wratchford\\Files" , NULL);
```

Как воспользоваться волшебной силой оболочки

Функции оболочки занимают видное место в программировании под управлением Windows 98/2000. Более новые *Sh*ell-функции больше известны как SH*x*-функции, поскольку многие из них имеют приставку SH. Некоторые из тех, что поддерживаются в Windows 2000 (как старые, так и новые), описаны в следующих разделах. Несколько старых Win32 API-функций были заменены новыми Shell-функциями, что объясняется их универсальностью. Старые API-функции по-прежнему используются для простых передач или манипуляций, но их новые “коллеги” работают быстрее, хотя порой не слишком просты в применении. Ну да ладно, вряд ли вам приходилось слышать, что программирование в Windows — простое занятие, разве что шутки ради.

Стыковка с броузером

При таком большом объеме данных, доступных сегодня через World Wide Web, часто возникает необходимость во взаимодействии приложения с системным Web-броузером. Shell-расширения Win32 API в состоянии обеспечить и этот тип интерфейса. В следующем примере демонстрируется процедура, которая использует функцию `ShellExecute()`, чтобы открыть броузер с заданным адресом (URL). Функция `ShellExecute()` представляет собой сокращенную версию описанной выше функции `ShellExecuteEx()`.

```
int  RunBrowser (char* URL)
{
    if (! ShellExecute(NULL, "open", URL, NULL, NULL, SW_SHOW))
    {
        char data[100];
        sprintf(data, "Could not run browser with URL '%s'",URL);
        MessageBox(NULL, data, "Operation Error!", MB_OK);
        return -1;
    }
    return 0;
}
```

Второй параметр (`"open"`) означает намерение открыть файл. Третий параметр предоставляет информацию о местоположении файла. С помощью библиотеки Shell выполняется дешифровка URL-дескрипторов, а затем используется зарегистрированный в системе броузер, который работает с URL-адресами.

Поддержка более сложных операций с файлами

Функция `SHFileOperation()` отличается универсальностью выполнения операций с файлами, что зачастую вызывает определенные трудности у начинающих, но зато она пользуется большим спросом благодаря своей способности поддерживать более сложные передачи данных. Функция `SHFileOperation()` значительно превосходит по мощности функцию `MoveFile()`, поскольку она может передавать файлы и папки на новый носитель (том) информации. Она способна также перемещать дочерние файлы внутри папки. Функция `SHFileOperation()` широко используется в Windows 95 и более поздних версиях Windows. При своей мощности она довольно гибка, и ее часто используют при перемещении папок и опустошении корзины (Recycle Bin).

В примере программы, приведенной в листинге 24.5 и представленной на компакт-диске, прилагаемом к этой книге, показано, как копировать сразу все файлы и подпапки из одного места в другое.

Листинг 24.5. Использование функции `SHFileOperation()`

```
// Объявляем структуру SHFILEOPSTRUCT, чтобы заполнить ее
// информацией для использования функции SHFileOperation.
SHFILEOPSTRUCT op;

// В целях безопасности очищаем область, занимаемую структурой.
  ZeroMemory(&op, sizeof(op));

  String RestoreDir, RestoreToDir;
```

```
/* Назначим строковой переменной RestoreDir папку источника. */

     RestoreDir = "C:\\MYFILES\\";

// Устанавливаем длину строки и прибавляем 1.
     RestoreDir.SetLength(RestoreDir.Length() + 1);

// Добавляем NULL-символ.
     RestoreDir[RestoreDir.Length()] = '\0';
// Проделаем это снова. Инкрементируем строку и добавляем другую.
     RestoreDir.SetLength(RestoreDir.Length() + 1);
     RestoreDir[RestoreDir.Length()] = '\0';

// Строка RestoreToDir будет содержать папку приемника, в которую
// будут скопированы файлы.
   RestoreToDir = "C:\\TEMP"

// Добавляем "символ" пути и два NULL-символа.
      RestoreToDir  = RestoreToDir +  "\\";
      RestoreToDir.SetLength(RestoreToDir.Length()+ NULL + NULL);

op.hwnd = 0;     // Это дескриптор для главного окна, равен 0.

op.wFunc= FO_COPY;    // Предписываем функции SHFileOperation
                      // копировать файлы.

op.pFrom = RestoreDir.c_str();  // Папка источника.

op.pTo =  RestoreToDir.c_str();   // Папка приемника.

op.fFlags=FOF_FILESONLY;   // Предписываем копировать только
                           // файлы, даже если задано *.*.

/* Здесь мы назначаем целое значение литералу 'copy_done'.
 Поскольку функция SHFileOperation возвращает значение, можно
 перехватить его для обработки ошибки. В этой части для сообщений
 мы будем использовать надписи (элемент Label). */

//Выполняем функцию SHFileOperation и перехватываем значение.
    int copy_done = SHFileOperation(&op);

    if (copy_done == 0)
    {
      if (op.fAnyOperationsAborted)
       {
         Label1->Caption = "Процесс копирования остановлен!";
       }
      else
        {
         Label1->Caption =
```

```
                    "Операция восстановления прошла успешно!";
        }
    }
  else
    {
      Label1->Caption = "Процесс копирования не удался.";
    }
```

При выполнении этого примера все файлы и подпапки, входящие в состав папки MYFILES, будут скопированы в папку \TEMP.

Как показано в следующем фрагменте программы, функция SHFileOperation() требует, чтобы строка RestoreDir оканчивалась двумя NULL-символами. Несоблюдение этого требования часто вызывает проблемы.

```
// Добавляем NULL.
     RestoreDir[RestoreDir.Length()] = '\0';
// Проделаем это снова. Инкрементируем строку и добавляем другую.
     RestoreDir.SetLength(RestoreDir.Length() + 1);
     RestoreDir[RestoreDir.Length()] = '\0';
```

Как показано в листинге 24.5, прежде чем вызывать функцию SHFileOperation(), необходимо инициализировать структуру op типа SHFILEOPSTRUCT, которая содержит множество элементов и имеет следующий формат.

```
typedef struct _SHFILEOPSTRUCT {  // shfos
    HWND         hwnd;
    UINT         wFunc;
    LPCSTR       pFrom;
    LPCSTR       pTo;
    FILEOP_FLAGS fFlags;
    BOOL         fAnyOperationsAborted;
    LPVOID       hNameMappings;
    LPCSTR       lpszProgressTitle;
}  SHFILEOPSTRUCT, FAR *LPSHFILEOPSTRUCT;
```

На первый взгляд кажется, что слишком уж много элементов нужно заполнить, чтобы получить пользу от этой структуры.

hwnd — дескриптор диалогового окна, используемого для отображения информации о состоянии выполнения операции. Вам уже должен быть знаком этот важный атрибут.

Поле wFunc определяет выполняемую операцию. Этот член может иметь одно из значений, перечисленных в табл. 24.17.

Таблица 24.17. Возможные операции функции SHFileOperation()

Операция	Описание
FO_COPY	Копирует файлы, заданные полем pFrom в область, заданную полем pTo. Может копировать файлы из одного места в другое независимо от принадлежности к тому (логическому разделу на диске, создаваемому при его форматировании)
FO_DELETE	Удаляет файлы, заданные полем pFrom. Поле pTo игнорируется, поскольку мы просто удаляем файлы. См. флаги, относящиеся к параметру для операций с корзиной (Recycle Bin)

Окончание табл. 24.17

Операция	Описание
FO_MOVE	Перемещает файлы, заданные полем pFrom, в область, заданную полем pTo. При этом действительно происходит перемещение исходных файлов в другое место. Здесь также поддерживаются передачи между томами
FO_RENAME	Переименовывает файлы, заданные полем pFrom

Поле pFrom представляет указатель на буфер, задающий один или несколько имен исходных файлов. Имена (если их несколько) должны быть разделены null-символами. Список имен должен завершаться двумя null-символами. Как упоминалось выше, необходимо соблюдать следующее правило. Например, если у вас в списке много файлов, в качестве разделителя их имен используйте символ \0, а в конце списка вставьте этот символ дважды.

Поле pTo представляет указатель на буфер, который содержит имя файла или папки приемника. Этот буфер может включать несколько имен файлов, если член структуры fFlags содержит значение FOF_MULTIDESTFILES. Имена (если их несколько) должны быть разделены null-символами. Список имен должен завершаться двумя null-символами.

Поле fFlags означает признаки (флаги), которые управляют файловыми операциями. Этот член может представлять собой комбинацию значений, перечисленных в табл. 24.18.

Таблица 24.18. Флаги операций над файлами

Флаг	Значение
FOF_ALLOWUNDO	Сохраняет информацию для операции отмены, если таковая возможна
FOF_CONFIRMMOUSE	Не реализован
FOF_FILESONLY	Выполняет операцию только с файлами, если задано шаблонное имя файла (*.*)
FOF_MULTIDESTFILES	Означает, что член pTo задает не одну папку, в которую были бы помещены все исходные файлы, а несколько файлов приемника (по одному для каждого файла источника)
FOF_NOCONFIRMATION	При ответе используется вариант Yes to All (Да для всех) в любом отображаемом диалоговом окне
FOF_NOCONFIRMMKDIR	Не подтверждает создание новой папки, если этого требует операция
FOF_RENAMEONCOLLISION	Присваивает новое имя (такое как Copy #1 of...) файлу, задействованному в операции перемещения, копирования или переименования, если файл с заданным именем приемника уже существует
FOF_SILENT	Не отображает диалоговое окно состояния. Пока функция находится в состоянии выполнения операции, ничего отображаться не будет
FOF_SIMPLEPROGRESS	Отображает диалоговое окно состояния, но не сообщает имена файлов
FOF_WANTMAPPINGHANDLE	Заполняет член hNameMappings. Этот дескриптор должен освобождаться с помощью функции SHFreeNameMappings()

Член fAnyOperationsAborted используется для обозначения результата завершения операции. Значение TRUE используется в том случае, если пользователь прервал файловую операцию до ее завершения; значение FALSE означает успешно завершившуюся операцию.

Член hNameMappings идентифицирует дескриптор объекта преобразования имени файла, содержащего массив структур типа SHNAMEMAPPING. Каждый элемент структуры этого массива содержит старый и новый пути для каждого файла, который был перемещен, скопирован или переименован. Этот член используется только в том случае, если поле fFlags содержит значение FOF_WANTMAPPINGHANDLE.

Член lpszProgressTitle представляет указатель на строку, предназначенную для названия диалогового окна. Этот член используется только в том случае, если поле fFlags содержит значение FOF_SIMPLEPROGRESS. Этот член позволяет подобрать заголовок диалогового окна для вашего приложения, т.е. вместо такого текста в заголовке, как Deleting (Удаление) или Copying Files... (Копирование файлов...) вы можете установить собственный заголовок. Например, вы могли бы в заголовке дать более подробную информацию: Происходит удаление файлов господина Петренко....

В листинге 24.5 сначала объявляется переменная записи op типа SHFILEOPSTRUCT. Объявив ее, лучше всего очистить занимаемую ею область памяти с помощью функции ZeroMemory(). После обнуления структуры можно приступать к ее использованию. Существует множество способов инициализации переменной записи типа SHFILEOPSTRUCT, которые способны повлиять на выполнение функции SHFileOperation().

Двумя весьма важными элементами записи типа SHFILEOPSTRUCT являются члены pTo и pFrom. При заполнении этой информации не забудьте при необходимости завершить указанные строки двумя null-символами. Наличие только одного null-символа заставит функцию SHFileOperation() "считать", что в представленном вами списке файлов содержится лишь один элемент, в результате чего будет выполнена только одна операция для этого одного элемента. А для того чтобы "заработал" список элементов, т.е. чтобы функция SHFileOperation() действовала согласно списку, необходимо завершить нужные строки двумя null-символами. Однако и этого недостаточно. Для корректного выполнения функции в соответствии с вашими намерениями важно использовать флаг FOF_MULTIDESTFILES. В противном случае будет сгенерирован признак ошибки.

После объявления структуры типа SHFILEOPSTRUCT, ее очистки и заполнения ее членов, можно вызывать функцию SHFileOperation().

В листинге 24.5 ясно показано, как работает функция SHFileOperation(). Эта функция поддерживается только в Windows 95/98/Millennium (Me) и NT 2000.

Функция SHFileOperation() используется также для удаления файлов и размещения их в корзине (Recycle Bin). Ниже приведен пример окна списка DirectoryListBox. Внутри него пользователю предоставляется возможность выбрать папку и удалить ее. В листинге 24.6 показано, как удаляется папка.

Листинг 24.6. Удаление всех файлов в папке

```
SHFILEOPSTRUCT op;  // Объявляем переменную структуры.

// Очищаем память.
  ZeroMemory(&op, sizeof(op));
  String DelDir;    // Дескриптор папки.

  // Получаем выделенный ТЕКУЩИЙ элемент в списке...
   DelDir =
 DirectoryListBox2->Items->Strings[DirectoryListBox2->ItemIndex];

        DelDir.SetLength(DelDir.Length() + 1);
```

```
            DelDir[DelDir.Length()] = '\0';
       DelDir.SetLength(DelDir.Length() + 1);
            DelDir[DelDir.Length()] = '\0';

  String BS;
  BS =  "Вы собираетесь УДАЛИТЬ " + DelDir +
                                              "!!! Вы уверены в этом?";
  int theanswer = Application->MessageBox(BS.c_str(),
                                                  WARNING!, MB_YESNO);

   switch(theanswer)
    {
     case ID_NO:
                 if(NoNag==0)
                  { ShowMessage(
                     "Не торопитесь, подумайте еще!");}  return;
    }
    // Устанавливаем члены структуры (источник и приемник) для
    // функции SHFIleOperation...
    op.hwnd = 0;
    op.wFunc= FO_DELETE;     // Флаг удаления файлов.
    op.pFrom = DelDir.c_str();
    op.fFlags=FOF_ALLOWUNDO;
// Флаг отправки в корзину (должен иметь значение FO_DELETE).

    int copy_done = SHFileOperation(&op);

    if (copy_done == 0)
    {
      if (op.fAnyOperationsAborted)
        {
         ShowMessage("Вы прервали удаление папки. Некоторые файлы
                     могут содержаться в корзине. Загляните туда,
                     если вам нужно восстановить эти файлы.");
        }
      else
        {
         ShowMessage("Удаление папки прошло успешно!");
        }
    }
  else
    {
       ShowMessage("Удаление папки оказалось неудачным.");
    }  // Боже! Сколько работы!

// Поскольку мы удалили папку в списке DirectoryListBox,
// необходимо скорректировать его, чтобы
// он не содержал несуществующего элемента!
 DirectoryListBox2->Items->Strings[DirectoryListBox2->ItemIndex-1];
```

```
// Обновляем DirectoryListbox
 DirectoryListBox2->Update();
```

При выполнении этой программы происходит переход в родительскую папку, выбранную пользователем, и удаляются все файлы и дочерние папки внутри родительской папки. Все файлы перемещаются в корзину. Чтобы уведомить функцию `SHFileOperation()` о вашем намерении, важно использовать флаг `FOF_ALLOWUNDO` для атрибута `fFlags` в сочетании с флагом `FO_DELETE` для атрибута `wFunc`.

На этих двух примерах была продемонстрирована мощь `SHx`-функций. Но это не предел! В "пороховницах" других `SHx`-функций есть и другой "порох".

Операции с корзиной

Библиотека функций системной оболочки предоставляет возможность выполнять такие запросы к корзине (Recycle Bin), как определение количества содержащихся в ней файлов и ее размера в байтах, а также производить некоторые действия, например опустошение корзины без вмешательства пользователя. Библиотекой Shell DLL (Windows 98/2000) поддерживаются новые функции `SHEmptyRecycleBin()` и `SHQueryRecycleBin()`, используемые для манипуляций с корзиной. Например, в следующем фрагменте программы запрашивается количество элементов в корзине, которая затем опустошается. Пользователь вообще не должен "заглядывать" в корзину. Подобные действия могут выполняться в системной утилите очистки от мусора и в других целях.

```
// Объявляем переменную структуры.
SHQUERYRBINFO RCinfo;
 RCinfo.cbSize = sizeof(shqbi);
 // Общий размер объектов в корзине, выраженный в байтах,
 // установим  пока равным нулю
 RCinfo.i64Size = 0;
 // Общее количество элементов в корзине.
 RCinfo.i64NumItems  = 0;


// Узнаем, сколько в корзине элементов, и занесем это
// значение в структуру.
  if(S_OK != SHQueryRecycleBin("C:\ \ ", &shqbi) )
   {
    Label1->Caption = "Ошибка!";
     return;
   }

   int one = RCinfo.i64NumItems;
   int two = RCinfo.i64Size;

// Теперь удалим все элементы из корзины. Для этого нужно
// подтверждение пользователя!
 SHEmptyRecycleBin(0,0,SHERB_NOPROGRESSUI);
   Label1->Caption = one;
   Label2->Caption = two;
```

Эта функция также поддерживается в Windows 95, но для этого у вас должна быть библиотека `Shell32.dll` версии 4.00.

Структура SHQUERYRBINFO невелика по размеру. Ее формат имеет следующий вид.

```
typedef struct _SHQUERYRBINFO {
    DWORD cbSize;
    __int64 i64Size;
    __int64 i64NumItems;
}  SHQUERYRBINFO, *LPSHQUERYRBINFO;
```

Член cbSize содержит общий размер структуры.

Член i64Size предназначен для хранения общего размера всех объектов (в байтах) заданной корзины.

Член i64NumItems представляет общее количество элементов в корзине.

Две новых "корзинных" API-функции, SHEmptyRecycleBin() и SHQueryRecycleBin(), не требуют много параметров (они описаны в следующих разделах).

SHQueryRecycleBin()

Функция SHQueryRecycleBin() формирует запрос к корзине. Она может возвращать ее элементы, их количество и размер.

```
HRESULT SHQueryRecycleBin (
    LPCTSTR pszRootPAth,
    LPSHQUERYRBINFO pSHQueryRBInfo
);
```

В случае успешного выполнения эта функция возвращает только одно "благополучное" значение (S_OK), в противном случае она может возвратить признак ошибки.

Первый параметр, pszRootPath, идентифицирует строку с завершающим null-символом, которая содержит путь к корзине, обязательно включающий спецификатор устройства и разделитель пути. Можно также задать папки (или каталоги), например c:\\Leroy Burfict PC\\MyFiles\\Work. При задании значения NULL или 0 функция прочитает информацию о корзинах на всех устройствах.

Второй параметр, pSHQueryRBInfo, идентифицирует адрес структуры SHQUERYRBINFO, которая содержит информацию о корзине. Ее член cbSize должен быть установлен до вызова этой функции.

SHEmptyRecycleBin()

Функция SHEmptyRecycleBin() используется для опустошения корзины.

```
HRESULT SHEmptyRecycleBin (
    HWND hwnd,
    LPCTSTR pszRootPAth,
     DWORD dwFlags
   );
```

В случае успешного выполнения эта функция возвращает только одно "благополучное" значение (S_OK), в противном случае она может возвратить признак ошибки. Возвращаемое значение этой функции совпадает со значением, возвращаемым функцией SHQueryRecycleBin().

Первый параметр, hwnd, идентифицирует дескриптор для родительского окна диалогового окна или другого окна, которое может быть отображено во время выполнения операции. Если диалоговое или другое окно не отображается, этот параметр может быть равным значению NULL.

Второй параметр, pszRootPath, совпадает с описанным выше для функции SHQueryRecycleBin(). Он идентифицирует указатель на строку с завершающим null-символом, которая содержит путь к корзине, обязательно включающий спецификатор

устройства и разделитель пути. Можно также задать папки (или каталоги), например `c:\\Will Carter PC\\Pics\\Pub`. При задании значения `NULL` или `0` функция прочитает информацию о корзинах на всех устройствах.

Третий параметр, `dwFlags`, используется для идентификации одного или нескольких значений, перечисленных в табл. 24.19.

Таблица 24.19. Флаги функции `SHEmptyRecycleBin`

Флаг	Значение
`SHERB_NOCONFIRMATION`	Не отображается диалоговое окно для подтверждения удаления объектов
`SHERB_NOPROGRESSUI`	Не отображается диалоговое окно для индикации степени выполнения функции
`SHERB_NOSOUND`	По завершении операции звуковые сигналы не воспроизводятся

Просмотр папок

Еще одной Shell-функцией, заслуживающей внимания программистов, является функция `SHBrowseForFolder()`. Она проводит обзор папок и дает возможность пользователю выбрать одну из них. Вы уже, вероятно, познакомились с работой этой функции при инсталляции устройств и других Windows-запросов, связанных с поиском местонахождения файлов.

Предоставление возможности пользователю выбрать местоположение папки может быть чрезвычайно полезным. Чтобы понять, как использовать функцию `SHBrowseForFolder()`, давайте рассмотрим пример, позволяющий пользователю выбрать существующую папку. Местонахождение папки будет затем помещено в поле приемника, которое можно применить для вложения файлов и подпапок.

Поиск папки

Запустите C++Builder и поместите в форму кнопку и надпись, причем их взаимное расположение значения не имеет.

Вставьте в начало кода модуля `Unit1.cpp` следующее объявление заголовочного файла и строковой переменной.

```
#define NO_WIN32_LEAN_AND_MEAN
#include <shlobj.h>
String Directory;
```

Используя окно Object Inspector, создайте событие `OnClick` для кнопки, помещенной в форму. В метод обработки этого события вставьте следующий код. Скомпилируйте и запустите приложение.

```
BROWSEINFO BrowsingInfo;
    char FolderName[MAX_PATH];
    LPITEMIDLIST ItemID;

    memset(&BrowsingInfo, 0, sizeof(BROWSEINFO));
    BrowsingInfo.hwndOwner      = Handle;
    BrowsingInfo.pszDisplayName = FolderName;
    BrowsingInfo.lpszTitle      = "Choose your folder:";
    ItemID = SHBrowseForFolder(&BrowsingInfo);
    if(ItemID) {
```

```
        char DirPath[MAX_PATH]= "";
        SHGetPathFromIDList(ItemID, DirPath);
        /* Путь сейчас содержится в переменной DirPath. */
    Directory = DirPath;
    }

    Label1->Caption = Directory;
```

При выполнении этого примера вы сможете выбрать папку (или каталог). Надпись будет содержать выбранную вами папку.

Функция `SHBrowseForFolder()` имеет только один параметр.

```
WINSHELLAPI LPITEMIDLIST WINAPI SHBrowseForFolder(
    LPBROWSEINFO lpbi
   );
```

Этот параметр представляет структуру типа `LPBROWSEINFO`. Функция возвратит указатель на идентификатор элемента в списке. Если пользователь решить отменить выбор (т.е. щелкнет на кнопке Cancel), функция возвратит значение `NULL`.

Переменная `lpbi` — указатель на структуру `LPBROWSEINFO`.

Структура содержит элементы, которые должны быть заполнены.

```
typedef struct _browseinfo {
    HWND hwndOwner;
    LPCITEMIDLIST pidlRoot;
    LPSTR pszDisplayName;
    LPCSTR lpszTitle;
    UINT ulFlags;
    BFFCALLBACK lpfn;
    LPARAM lParam;
    int iImage;
}  BROWSEINFO, *PBROWSEINFO, *LPBROWSEINFO;
```

Член `hwndOwner` идентифицирует дескриптор окна владельца для диалогового окна.

Член `pidlRoot` представляет указатель на список идентификаторов элементов (структуру `ITEMIDLIST`), задающий местонахождение корневой папки, обзор которой будет выполнен. В диалоговом окне отображаются только заданная папка и ее подпапки. Этот член может быть равен значению `NULL`; в этом случае используется корневая папка пространства имен (папка рабочего стола).

Структура `ITEMIDLIST` определяется следующим образом.

```
typedef struct _ITEMIDLIST {
    SHITEMID mkid;  // Список идентификаторов элементов.
}  ITEMIDLIST, * LPITEMIDLIST;
typedef const ITEMIDLIST * LPCITEMIDLIST;
```

Член `pszDisplayName` представляет указатель на буфер, который принимает имя папки, выбранное пользователем. Предполагается, что размер этого буфера равен `MAX_PATH` байт.

Член `lpszTitle` представляет указатель на строку с завершающим null-символом, которая отображается над деревом в диалоговом окне. Эту строку можно использовать для задания инструкций для пользователя.

Член `ulFlags` идентифицирует значение, задающее типы папок, которые могут быть перечислены в диалоговом окне, и другие опции. Этот член может включать некоторое количество значений (оно может быть равно нулю), которые перечислены в табл. 24.20.

Таблица 24.20. Типы папок

Флаг	Значение
BIF_BROWSEFORCOMPUTER	Возвращает только компьютерные папки. Если пользователь выберет вариант, отличный от компьютера, кнопка OK будет недоступна
BIF_BROWSEFORPRINTER	Возвращает только информацию о принтерах. Если пользователь выберет вариант, отличный от принтера, кнопка OK будет недоступна
BIF_DONTGOBELOWDOMAIN	Не включает сетевые папки ниже уровня домена в изображении дерева
BIF_RETURNFSANCESTORS	Возвращает только предков файловых систем. Если пользователь выберет вариант, отличный от предка файловой системы, кнопка OK будет недоступна
BIF_RETURNONLYFSDIRS	Возвращает только папки файловой системы. Если пользователь выберет папки, которые не являются частью файловой системы, кнопка OK будет недоступна
BIF_STATUSTEXT	Включает область состояния в диалоговом окне. Функция обратного вызова может устанавливать текст состояния путем отправки сообщений в диалоговое окно

Член `lpfn` идентифицирует адрес функции, определяемой приложением, которую вызывает диалоговое окно при возникновении событий. Этот член может быть равен значению `NULL`. Подробнее см. описание функции `BrowseCallbackProc()` в справочной документации по Win32 API.

Член `lParam` представляет определяемое приложением значение, которое диалоговое окно передает функции обратного вызова (если таковая задана).

Член `iImage` идентифицирует переменную, которая принимает изображение, связанное с выбранной папкой. Изображение задается как индекс для списка системных изображений.

Как видите, описанные нами стандартные Shell- и `SHx`-функции обладают определенной гибкостью. Тема Shell-программирования настолько обширна, что ей можно было бы посвятить целую книгу. За получением более подробной информации мы рекомендуем обратиться к справочному руководству по интерфейсу Win32 API C++Builder (Win32 API Reference Guide), а также вам не помешает поэкспериментировать с несколькими недокументированными средствами оболочки, включенными в состав библиотеки `SHELL32.DLL`, но при этом не забывать о мерах предосторожности. С магией оболочки (как и с любой магией) шутки плохи.

Реализация мультимедийных служб

Как описано выше, Win32 обеспечивает работу немалого числа мультимедийных служб. Например, весьма полезно иметь возможность воспроизводить аудио- или видеоданные и поддерживать точность таймеров.

Воспроизведение мультимедийных файлов

Borland предоставляет VCL-компонент `TMediaPlayer`, который можно использовать для манипуляции и воспроизведения мультимедийных клипов. Компонент `TMediaPlayer` обеспечивает оболочку для MCIWnd-функций, поддерживаемых библиотекой `VFW32.DLL`. Несмотря на всю полезность компонента `TMediaPlayer`, непосредственный вызов MCIWnd-функций может обеспечить б*о*льшую гибкость, а простота их применения еще

более увеличивает их ценность. Например, функцию MCIWndCreate() можно использовать для воспроизведения музыки, записанной на компакт-дисках, аудиофайлов (wave-файлов), MIDI-файлов или видеоклипов (рис. 24.3). Полный текст программы, описанной в этом разделе, можно найти в папке MMedia компакт-диска, прилагаемого к этой книге.

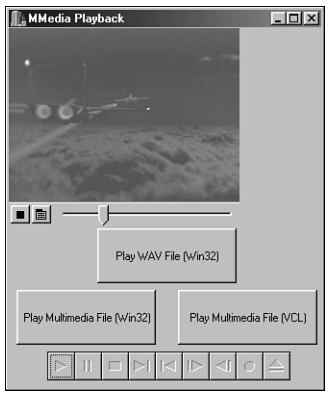


Рис. 24.3. Программа воспроизведения мультимедийной информации

В следующем фрагменте программы показано, как мультимедийные возможности (в виде Win32 API-функций) можно использовать для воспроизведения видеоклипа.

```
void __fastcall TForm1::SpeedButtonPlayMMFileUsingWin32Click(TObject *Sender)
{
  OpenDialog1->DefaultExt = "AVI";
  OpenDialog1->FileName = "*.avi";
  if (OpenDialog1->Execute())
    MCIWndCreate(Handle,          // Дескриптор окна приложения.
        NULL,                     // Дескриптор экземпляра.
        WS_VISIBLE | WS_CHILD | MCIWNDF_SHOWALL,  // Стили окна.
        OpenDialog1->FileName.c_str());           // Имя файла.
}
```

Функция MCIWndCreate() создает (используя MCIWND_WINDOW_CLASS) VCR-подобное окно (Video Cassette Recorder — кассетный видеомагнитофон), содержащее кнопку Play/Stop и панель для отображения видеоклипов.

Однако для звуковых файлов проще использовать функцию PlaySound().

```
void __fastcall TForm1::SpeedButtonPlayWaveUsingWin32Click(TObject *Sender)
{
  OpenDialog1->DefaultExt = "WAV";
  OpenDialog1->FileName = "*.wav";
  if (OpenDialog1->Execute())
    PlaySound(OpenDialog->FileName.c_str(), NULL, SND_ASYNC);
}
```

PlaySound() — чрезвычайно полезная функция, предоставляемая библиотекой mmsystem.dll. Часто для создания звукового эффекта возникает потребность воспроизводить звуковые файлы в фоновом режиме приложения. Функция PlaySound() не предоставляет VCR-подобных элементов управления, как в случае использования функции MCIWndCreate(). Более того, звуки воспроизводятся моментально при вызове функции PlaySound(). Средства DirectSound, поддерживаемые интерфейсом DirectX, предоставляют дополнительные возможности для манипуляции звуковыми данными и позволяют управлять несколькими источниками звуковой информации, воспроизводимой одновременно. Однако рассмотрение средств DirectSound выходит за рамки этой главы.

Как добиться точности швейцарских часов с помощью мультимедийного таймера

Borland предоставляет очень простой в применении VCL-компонент таймера, известный под именем `TTimer`. Компонент `TTimer` инкапсулирует таймерные функции интерфейса Win32 API. Он использует функцию `SetTimer()`, чтобы разрешить возникновение событий таймера, обеспечивает обработчик события `OnTimer` для ответа на уведомления о сообщениях `WM_TIMER` и использует функцию `KillTimer()` для запрещения событий таймера. При вызове функции `SetTimer()` приложение "просит" систему Windows уведомлять процесс (приложение) о постоянных обновлениях до тех пор, пока таймер не будет запрещен с помощью функции `KillTimer()`. Частота этих обновлений основана на интервале, заданном параметром функции `SetTimer()` (или значением свойства `Interval` компонента `TTimer`). Хотя интервал можно задать в миллисекундах, таймеры не всегда столь точны. Уведомления о сообщениях `WM_TIMER` могут происходить чаще, чем ожидается, или, наоборот, реже, если другие процессы задерживают систему. Иными словами, их точность оставляет желать лучшего. Однако, используя мультимедийные таймеры, можно получать более точные обновления. Мультимедийный таймер — это расширение оригинального интерфейса Win32 API, которое обеспечивает более высокое разрешение, чем стандартный таймер Windows.

При рассмотрении файла `mmsystem.h` ("совместное производство" Microsoft и Borland) можно ближе познакомиться с интересующими нас функциями.

```
/* Прототипы функций таймера.*/
WINMMAPI MMRESULT WINAPI timeGetSystemTime(LPMMTIME pmmt, UINT cbmmt);
WINMMAPI DWORD WINAPI timeGetTime(void);
WINMMAPI MMRESULT WINAPI timeSetEvent(UINT uDelay, UINT uResolution,
    LPTIMECALLBACK fptc, DWORD dwUser, UINT fuEvent);
WINMMAPI MMRESULT WINAPI timeKillEvent(UINT uTimerID);
WINMMAPI MMRESULT WINAPI timeGetDevCaps(LPTIMECAPS ptc, UINT cbtc);
WINMMAPI MMRESULT WINAPI timeBeginPeriod(UINT uPeriod);
WINMMAPI MMRESULT WINAPI timeEndPeriod(UINT uPeriod);
```

В большинстве случаев функций `timeSetEvent()` и `timeKillEvent()` вполне достаточно, чтобы использовать мультимедийный таймер. Для обработки уведомлений о сообщениях мультимедийного таймера внутри программы необходимо определить сообщение обратного вызова. В листинге 24.7 приводится пример приложения, в котором стандартный VCL-таймер сравнивается с мультимедийным таймером (эту программу можно найти в папке `MMTimer` на компакт-диске, прилагаемом к этой книге). На рис. 24.4 это приложение показано в действии: вы сами можете убедиться в существенной разнице между истекшим временем, отсчитанным двумя разными таймерами.

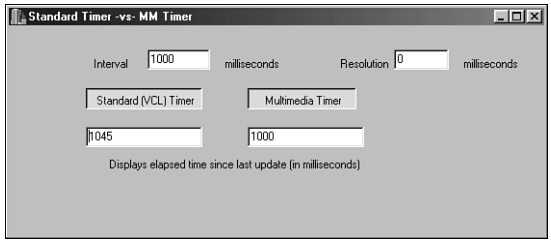


Рис. 24.4. Разница в точности между VCL- и мультимедийным таймером

Листинг 24.7. Код работы таймера

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

#include "mmtimer.h"
#include <mmsystem.h>
#include <time.h>

//---------------------------------------------------------------
#pragma package(smart_init)
#pragma resource "*.dfm"
TForm1 *Form1;
//---------------------------------------------------------------
__fastcall TForm1::TForm1(TComponent* Owner)
        : TForm(Owner)
{
        MMTimerID = 0;
        MMlasttime = 0;
        STANlasttime = 0;
}
//---------------------------------------------------------------

void __fastcall TForm1::Timer1Timer(TObject *Sender)
{

        unsigned int clocktime = clock();
        EditStandardTimer->Text = AnsiString(clocktime - STANlasttime);
        STANlasttime = clocktime;
}
//---------------------------------------------------------------

void __fastcall TForm1::HandleMMTimerEvent()
{
        unsigned int clocktime = clock();
        EditMMTimer->Text = AnsiString(clocktime - MMlasttime);
        MMlasttime = clocktime;
        //Refresh();
}
//---------------------------------------------------------------

void CALLBACK TimerProc(unsigned int uID, unsigned int uMsg,
                                    DWORD dwUser, DWORD dw1, DWORD dw2)
{
        Form1->HandleMMTimerEvent();
}

//---------------------------------------------------------------
void __fastcall TForm1::SpeedButtonMMTimerClick(TObject *Sender)
{
 if (SpeedButtonMMTimer->Down)
```

```
 {
    int Interval = EditInterval->Text.ToIntDef(0);
    int Resolution = EditResolution->Text.ToIntDef(0);  // 0 =
                                       // максимально возможная точность
    if (Timer1->Interval > 0)
    {
            MMlasttime = clock();
            MMTimerID =  timeSetEvent(Interval, Resolution,
                                TimerProc,NULL, TIME_PERIODIC);
    }
 }
 else
 {
    timeKillEvent(MMTimerID);
    MMTimerID = 0;
 }
}
//---------------------------------------------------------------
void __fastcall TForm1::SpeedButtonStandardTimerClick(TObject *Sender)
{
   if (SpeedButtonStandardTimer->Down)
   {
        Timer1->Interval = EditInterval->Text.ToIntDef(0);
        STANlasttime = clock();
        if (Timer1->Interval > 0)
                Timer1->Enabled = true;
   }
   else
   {
        Timer1->Enabled = false;
   }

}
//---------------------------------------------------------------
void __fastcall TForm1::FormClose(TObject *Sender, TCloseAction &Action)
{
    if (MMTimerID) timeKillEvent(MMTimerID);
}
```

Использование глобально уникальных идентификаторов (GUID)

Microsoft обеспечивает поддержку генерации глобально уникального идентификатора (Globally Unique Identifier — GUID), который представляет собой абсолютно уникальное 128-разрядное число. Подобно отпечаткам пальцев, считается, что не существует двух одинаковых GUID-значений. Это делает GUID-значение чрезвычайно полезным для уникальной идентификации элементов данных в приложении и в среде распределенной сети. На самом деле GUID-число представляет собой структуру следующего вида.

```
typedef struct _GUID {
    DWORD Data1;
    WORD   Data2;
    WORD   Data3;
    BYTE  Data4[8];
}  GUID;
```

Элемент Data1 предоставляет весьма полезный 32-разрядный ключ, как продемонстрировано в следующей функции.

```
unsigned int GetNewID()
{
    GUID guid;
    ::CoCreateGuid(&guid);

    //unsigned int - целое без знака
    return guid.Data1;
}
```

Чтобы эта функция заработала, в исходный код программы необходимо включить заголовочный файл objbase.h или заголовочный файл, который наследует файл objbase.h. Для генерации 128-разрядного ключа используется Win32-функция CoCreateGuid(). Функция GetNewID() возвращает лишь 32-разрядный элемент Data1. Этот 32-разрядный элемент часто обеспечивает достаточную уникальность для идентификации элементов данных. Однако, чтобы гарантированно получить по-настоящему глобально уникальный идентификатор, предлагается использовать полное 128-разрядное значение. Эту нестандартную функцию можно найти в исходном файле win32_util.cpp, который находится в папке Win32Code компакт-диска, прилагаемого к данной книге.

Определение системной информации

Существует несколько функций Win API, которые можно использовать для считывания системной информации. Они позволяют определить имена пользователей (login names), имена компьютеров, объем доступной памяти и версию Windows. Ниже описаны некоторые нестандартные функции, которые обращаются к интерфейсу Win32 API для получения системной информации. Их прототипы содержатся в заголовочном файле WINBASE.H.

Регистрационное имя пользователя

GetUserName() — это функция Win32 API, которая используется для считывания имени текущего пользователя, зарегистрированного в системе. В качестве параметров передаются буфер для имени пользователя и указатель на размер этого буфера. Упомянутая функция Win32 API инкапсулирована внутри пользовательской функции LoginUserName(), которую можно найти в исходном файле win32_util.cpp, находящемся в папке Win32Code на прилагаемом к книге компакт-диске.

```
AnsiString LoginUserName()
{
    AnsiString Name;

    DWORD size = MAX_PATH+1;
    char name[MAX_PATH+1];
```

```
    name[0] = '\0';  // инициализация

    GetUserName(name, &size);

    Name = AnsiString(name);
    return Name;
}
```

Имя компьютера

Следующая пользовательская функция-контейнер используется для считывания имени компьютера, идентифицированного в системном реестре. Непосредственным получателем этой информации является функция Win32 API `GetComputerName()`. В качестве параметров передаются буфер для имени компьютера и указатель на размер этого буфера. Упомянутую пользовательскую функцию можно найти в исходном файле `win32_util.cpp` в папке `Win32Code` на прилагаемом к книге компакт-диске.

```
AnsiString ComputerName()
{
    AnsiString Name;
    DWORD size = MAX_COMPUTERNAME_LENGTH + 1;
    char name[MAX_COMPUTERNAME_LENGTH + 1];
    name[0] = '\0';  // инициализация

    GetComputerName(name, &size);

    Name = AnsiString (name);
    return Name;
}
```

Размер доступной памяти

Следующая пользовательская функция-контейнер используется для определения размера доступной памяти. Непосредственным получателем этой информации является функция Win32 API `GlobalMemoryStatus()`. После получения информации об объеме свободной памяти вычисляется размер в килобайтах и результат возвращается в виде значения типа `AnsiString`. Упомянутую пользовательскую функцию можно найти в исходном файле `win32_util.cpp` в папке `Win32Code` на прилагаемом к книге компакт-диске.

```
AnsiString MemFree()
{
  AnsiString FreeMem;
  MEMORYSTATUS memory ;

  memory.dwLength = sizeof (memory) ;
  GlobalMemoryStatus (&memory) ;

  unsigned int value2 = 0;
  unsigned int value1 = memory.dwAvailPhys / 1024;

  if (value1 >= 1000)
```

```
  {
     value2 = value1 / 1000;
     value1 = (value1 - (value2 * 1000.0));
     FreeMem = AnsiString(value2) + "," + AnsiString(value1) + " KB";
  }
  else
     FreeMem = AnsiString(value1)  + " KB";
  return FreeMem;
}
```

Местонахождение временных файлов

Иногда полезно хранить данные в такой временной "камере хранения", как папка Windows `Temp`. Однако папка `Temp` на разных компьютерах может иметь различный путь. К счастью, функция Win32 API `GetTempPath()` позволяет узнать путь папки `Temp`. В следующем примере показана пользовательская функция-контейнер, которая использует функцию `GetTempPath()`.

```
void _fastcall TempFileLocation(AnsiString& filelocation, AnsiString extension)
{
     AnsiString TempDir;
     UINT BufferSize = GetTempPath(0,NULL);
     TempDir.SetLength(BufferSize+1);
     GetTempPath(BufferSize+1,TempDir.c_str());
     unsigned int id = GetNewID();
     AnsiString temp = UINT_To_Ansi(id);
     AnsiString TempFile = AnsiString("temp") +
                                   UINT_To_Ansi(id) + "." + extension;
     char * tempfile = new char[TempDir.Length() + TempFile.Length()];
     sprintf(tempfile, "%s%s",TempDir.c_str(), TempFile.c_str());
     filelocation = AnsiString(tempfile);
     delete[] tempfile;
}
```

Этот метод используется для генерации имен папок, которые время от времени можно увидеть в папке `Temp` (особенно после прерванной инсталляции). В этом примере используется определенная выше функция-контейнер `GetNewId()` (см. раздел "Использование глобально уникальных идентификаторов (GUID)"). Функция `GetNewID()` считывает уникальное 32-разрядное значение, которое используется как часть имени файла, сгенерированного для временного файла. Упомянутые пользовательские функции можно найти в исходном файле `win32_util.cpp` в папке `Win32Code` на прилагаемом к книге компакт-диске.

Размер файлов

Приходилось ли вам когда-либо испытывать потребность в быстром получении размера файла? Если да, то для этого вам пригодится функция `CreateFile()`. Однако давайте начнем с примера, а все необходимые разъяснения последуют за ним.

Создайте новый проект приложения в среде самой последней (и самой могучей) версии C++Builder. Программа File Size относится к проекту `FSize.bpr`, который находится в папке `FSize` на прилагаемом к книге компакт-диске.

Используя следующую схему, поместите в форму Form1 компоненты и установите их свойства в соответствии с табл. 24.21. Компоненты FileListBox и DirectoryListBox находятся на вкладке Win3.1 палитры компонентов.

Таблица 24.21. Компоненты для помещения в форму

Компонент	Свойство
Form1	Height = 355 Width = 480
DirectoryListBox1	Height = 150 Left = 8 Width = 190 Top = 8 FileList = FileListBox1
FileListBox1	Height = 150 Left = 200 Width = 190 Top = 8 Mask = *.*
Label	Name = Label1 Caption = File Size Left = 8 Top = 175
Label	Name = Label2 Caption = 0000 Left = 55 Top = 175

Введите код, приведенный в листинге 24.8. Не забудьте включить заголовок stdlib.h для функции itoa().

Листинг 24.8. Определение размера файла

```
//---------------------------------------------------------------
#include <vcl.h>
#include <stdlib.h> // Не забудьте об этом включении для itoa()!
#pragma hdrstop

#include "Unit1.h"
//---------------------------------------------------------------
#pragma package(smart_init)
#pragma resource "*.dfm"
TForm1 *Form1;
//---------------------------------------------------------------
__fastcall TForm1::TForm1(TComponent* Owner)
        : TForm(Owner)
{
}
//---------------------------------------------------------------
```

```
char* CommaAdd(int nValue);
//---------------------------------------------------------------
char* CommaAdd(int nValue)
 {
  size_t i, j=0;
  char buff1[20];
  static char buff2[25];
// Преобразуем значение в строку ASCII.
   itoa(nValue , buff1, 10);

   int MEG;       // MEG: если размер > 999 999, используем
                  // мегабайты, в противном случае - килобайты...
    if(nValue > 999999)
    MEG = 1;
     else
      MEG = 0;

   // Вставляем запятые.
   for(i=0; i <strlen(buff1); i++)
    {
     if(i && ((strlen(buff1) -i) %3) == 0)
       {  buff2[i+j] = ',';  j++;   }
       buff2[i+j] = buff1[i];
    }
    buff2[i+j] = buff1[i];
    if(MEG == 1)
     strcat(buff2 , " M");
     else
      strcat(buff2 , " K");
    return buff2;
 }
//---------------------------------------------------------------

// Следующий код вставляем в обработчик события OnClick
// компонента FileListBox:
//---------------------------------------------------------------
void __fastcall TForm1::FileListBox1Click(TObject *Sender)
{
  HANDLE file;
     int size;
          AnsiString str;
 file = CreateFile(FileListBox1->FileName.c_str(),
                  GENERIC_READ,FILE_SHARE_READ,
                  NULL, OPEN_EXISTING, NULL, NULL);

  if(file == INVALID_HANDLE_VALUE)
    ShowMessage("Invalid handle!");
     else
         {
           size = GetFileSize(file, NULL);   str.SetLength(size);
```

```
            str = size;
                  char *GetFileSizeInfo;
                  GetFileSizeInfo = CommaAdd(size);
                  lb_Size->Caption = GetFileSizeInfo;
                }
}
```

Пользовательская функция `CommaAdd()` принимает целочисленное значение в качестве параметра и преобразует его в символьную строку `buff1`. В эту строку (буфер) вставляются запятые, причем их позиции зависят от размера (т.е. от того, в каких единицах будет он выражен: в мегабайтах или килобайтах), после чего функция возвращает подготовленный ею буфер.

Событие `OnClick` использует еще одну популярную функцию Win32 API `CreateFile()`. Эта функция не просто создает файл. Она может работать практически с любым потоком ввода/вывода. Например, переданный функции `CreateFile()` в качестве параметра флаг `GENERIC_READ` позволяет открыть файл только в режиме чтения.

В этом примере мы просто стремимся получить доступ к файлу, который идентифицирован внутри компонента `FileListBox`. Функция `CreateFile()` в качестве первого параметра принимает значение `FileListBox1->FileName.c_str()`. Это не что иное, как имя файла из окна списка. Обратите внимание на элемент `.c_str()` в конце этой строки. Он преобразует строку типа `AnsiString` в обычную переменную типа `char *`. Второй параметр, переданный функции `CreateFile()`, содержит значение `GENERIC_READ`, определяющее тип доступа, который необходимо использовать при создании файла. Флаг `GENERIC_READ` просто означает режим доступа только для чтения.

Поскольку возвращаемый дескриптор не полагается на унаследованные дочерние процессы или уровень безопасности, мы устанавливаем четвертый параметр равным значению `NULL`. Флаг `OPEN_EXISTING`, содержащийся в пятом параметре, означает, что если файл не существует, функция не будет выполнена. Наконец, шестой и седьмой параметры содержат два значения `NULL`. С помощью шестого параметра задаются атрибуты файла. Поскольку мы не работаем с ними, это значение установлено равным `NULL`. “Обнуление” седьмого параметра означает, что мы не создаем шаблонных файлов. В Windows 9x этот параметр должен всегда иметь значение `NULL`.

При успешном выполнении функция `CreateFile()` возвращает дескриптор файла. Если выполнение прошло неудачно, функция возвращает значение `INVALID_HANDLE_VALUE`. Мы создали сообщение, чтобы показать, что файл, возможно, уже используется. Однако с помощью функции Win32 API `GetError()` мы могли бы получить больше информации. Функция `CreateFile()` подробно рассматривалась в разделе “Базовые операции ввода/вывода файлов” этой главы.

В этом примере также участвует функция `ShowMessage()`. Эта очень полезная функция C++Builder используется для быстрого создания окна сообщения для пользователя.

Если все прошло без осложнений, вызывается функция `GetFileSize()`, которая возвращает размер файла в виде целого значения. Функция `GetFileSize()` объявляется следующим образом.

```
DWORD GetFileSize(
  HANDLE hFile,           // Дескриптор для файла.
  LPDWORD lpFileSizeHigh  // Старшее слово размера файла.
);
```

Функция `GetFileSize()` принимает дескриптор файла; в данном случае это как раз тот дескриптор, который вернула функция `CreateFile()`.

Второй параметр не имеет большого значения в нашем примере, поскольку мы хотим знать истинный размер или младшее слово размера файла. Если вам нужно получить старшее слово размера файла, вы могли бы вставить параметр, указывающий на значение типа DWORD (двойное слово). Чтобы получить представление о младшем и старшем словах размера файла, достаточно щелкнуть правой кнопкой мыши на имени файла в окне программы Windows Explorer и из контекстного меню выбрать команду Properties (Свойства). Вы увидите два значения — младшее и старшее слова, составляющие размер файла.

Затем объявляется переменная GetSizeInfo типа char *, и для получения значения GetSizeInfo вызывается описанная выше пользовательская функция CommaAdd(). Результат ее работы показан на рис. 24.5.

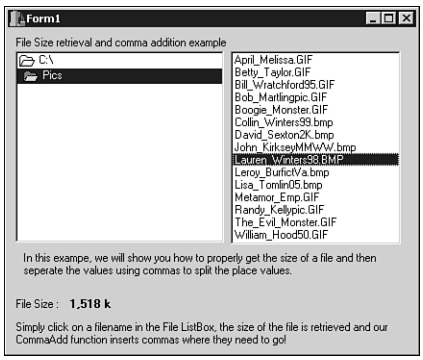


Рис. 24.5. Отображение размера файла с запятыми в соответствующих позициях

Свободное дисковое пространство и серийный номер

Теперь, когда мы знаем, как определить размер файла, давайте попробуем оценить объем доступного свободного пространства на диске. Помимо этой информации, иногда необходимо узнать серийный номер устройства памяти (диска). С такой задачей легко справляется API-функция GetVolumeInformation(), чего не скажешь об используемой ранее функции GetDiskFreeSpace(). И в самом деле, функция GetVolumeInformation() может считать всевозможную информацию об устройстве — имена томов, типы файловых систем и пр. В следующем примере мы скромно поинтересуемся лишь именем тома и серийным номером. Эти две API-функции стоит использовать совместно, и было бы неплохо объединить их в одну. К счастью, C++Builder упрощает их интеграцию.

Функция GetDiskFreeSpace() имеет следующие параметры.

```
BOOL GetDiskFreeSpace(

  LPCTSTR lpRootPathName,         // Адрес корневой папки.
  LPDWORD lpSectorsPerCluster,    // Адрес секторов в кластере.
  LPDWORD lpBytesPerSector,       // Адрес байтов в секторе.
  LPDWORD lpNumberOfFreeClusters, // Адрес числа свободных кластеров.
  LPDWORD lpTotalNumberOfClusters // Адрес общего числа кластеров.
);
```

Параметр lpRootPathName указывает на строку, содержащую спецификатор устройства и путь. Остальные параметры передаются для получения нужной информации.

- Параметр pSectorsPerCluster указывает на переменную для хранения количества секторов в кластере.
- Параметр lpBytesPerSector указывает на переменную для хранения количества байтов в секторе.
- Параметр lpNumberOfFreeClusters указывает на переменную для хранения количества свободных кластеров на диске.
- Параметр lpTotalNumberOfClusters указывает на переменную для хранения общего количества кластеров на диске.

В следующем примере мы также будем использовать функцию CommaAdd(), созданную в предыдущем примере. Она и в этом случае используется для форматирования числа путем

вставки запятых в соответствующие позиции. Наша цель — создать пользовательскую функцию GetDriveInfo().

Запустите C++Builder. Поместите в пустую форму шесть надписей и одну кнопку. Ни кнопка, ни надписи не должны иметь никаких имен. Затем в исходном заголовочном файле создайте пользовательское объявление в открытом (public) разделе класса, как показано в листинге 24.9.

Листинг 24.9. Пример объявления функции GetDriveInfo() (заголовочный файл)

```
//---------------------------------------------------------------
#ifndef Unit1H
#define Unit1H
//---------------------------------------------------------------
#include <Classes.hpp>
#include <Controls.hpp>
#include <StdCtrls.hpp>
#include <Forms.hpp>
//---------------------------------------------------------------
class TForm1 : public TForm
{
__published: // Компоненты, управляемые средой IDE.
        TButton *Button1;
        TLabel *Label1;
        TLabel *Label2;
        TLabel *Label3;
        TLabel *Label4;
        TLabel *Label5;
        TLabel *Label6;
        void __fastcall Button1Click(TObject *Sender);
private: // Объявления пользователя.
public:     // Объявления пользователя.
        __fastcall TForm1(TComponent* Owner);
        void __fastcall TForm1::GetDriveInfo (String &drive);

} ;
//---------------------------------------------------------------
extern PACKAGE TForm1 *Form1;
//---------------------------------------------------------------
#endif
```

После объявления пользовательской функции GetDriveInfo() в заголовочном файле осталось вставить следующий блок кода в исходный файл (с расширением .cpp), чтобы надлежащим образом объявить эту функцию.

```
void __fastcall TForm1::GetDriveInfo (String &drive)
{
  //Caption = String ("Details of drives on this system- ") + drive ;

  String YourDrive = drive;
  String volumeinfo;
  volumeinfo.SetLength (255);
```

```
// Устанавливаем длину, чтобы поместить спецификацию длины в
// функцию GetVolumeInformation()
  DWORD serialnumber ;

  if (GetVolumeInformation (YourDrive.c_str (),
                            volumeinfo.c_str(),
                            volumeinfo.Length(), &serialnumber,
                            NULL, NULL, NULL, NULL))
   {
    Label1->Caption = volumeinfo ;

    // Переводим целое значение в тип char для серийного номера
    char STRING[35];
    ltoa (serialnumber , STRING, 16);
    Label2->Caption = STRING;

    DWORD spc;        // Количество секторов в кластере.
    DWORD bps;        // Количество байт в кластере.
    DWORD cluster;    // Количество кластеров.
    DWORD freeclust;  // Количество свободных кластеров.

    GetDiskFreeSpace (YourDrive.c_str (),&spc,&bps,&freeclust,&cluster);
    Label3->Caption = CommaAdd(spc);
    Label4->Caption = CommaAdd(bps);
    Label5->Caption = CommaAdd(cluster);

    int free_bytes = freeclust * spc * bps;
    Label6->Caption = CommaAdd (free_bytes);
  }
  else
  {
    ShowMessage ("Problem reading drive information.");
  }
}
```

Мы создали эту функцию-контейнер, которая объединяет две функции Win32 API — `GetDiskFreeSpace()` и `GetVolumeInformation()` — на случай, если таковая понадобится вам в других приложениях. В ней используется функция `CommaAdd()` и интерфейс Win32 API, "стараниями" которых "добытая" системная информация выгружается в заблаговременно созданные нами поля для надписей (элементы управления надписями). Такая идея может пригодиться вам при создании окон About (О программе) и в функциях других типов. Теперь можно воспользоваться окном инспектора объектов Borland для создания обработчика события `OnClick` для нашей единственной кнопки. Вставьте следующий код в этой метод.

```
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  GetDriveInfo("C:\\");
}
```

Заголовочный файл этой программы должен быть аналогичен содержимому листинга 24.9, а `.cpp`-файл — содержимому листинга 24.10.

Листинг 24.10. Пример использования функции `GetDriveInfo()` (`.ccp`-файл)

```
//---------------------------------------------------------------
#include <vcl.h>
#include <stdlib.h>     // НЕ ЗАБУДЬТЕ включить этот заголовок
                        // для функции itoa()!
#pragma hdrstop

#include "Unit1.h"
//---------------------------------------------------------------
#pragma package(smart_init)
#pragma resource "*.dfm"
TForm1 *Form1;
char* CommaAdd(int nValue);
//---------------------------------------------------------------
__fastcall TForm1::TForm1(TComponent* Owner)
        : TForm(Owner)
{
}
//---------------------------------------------------------------
char* CommaAdd(int nValue)
 {
    size_t i, j=0;
  char buff1[20];
  static char buff2[25];
     // Преобразуем значение в строку ASCII.
   itoa(nValue , buff1, 10);

   // Вставляем запятые.
   for(i=0; i <strlen(buff1); i++)
    {
     if(i && ((strlen(buff1) -i) %3) == 0)
       {  buff2[i+j] = ',';  j++;   }
       buff2[i+j] = buff1[i];
    }
    buff2[i+j] = buff1[i];
    return buff2;

 }
//---------------------------------------------------------------
void __fastcall TForm1::GetDriveInfo (String &drive)
{
  //Caption = String ("Details of drives on this system- ") + drive ;

  String YourDrive = drive;
  String volumeinfo;
  volumeinfo.SetLength (255);
// Устанавливаем длину, чтобы поместить спецификацию длины в
// функцию GetVolumeInformation()
  DWORD serialnumber ;
```

```
  if (GetVolumeInformation (YourDrive.c_str (),
                                volumeinfo.c_str(),
                            volumeinfo.Length(), &serialnumber,
                            NULL, NULL, NULL, NULL))
   {
    Label1->Caption = volumeinfo ;

     // Переводим целое значение в тип char для серийного номера.
    char STRING[35];
    ltoa (serialnumber , STRING, 16);
    Label2->Caption = STRING;

    DWORD spc;          // Количество секторов в кластере.
    DWORD bps;          // Количество байт в кластере.
    DWORD cluster;      // Количество кластеров.
    DWORD freeclust;    // Количество свободных кластеров.

    GetDiskFreeSpace (YourDrive.c_str (),&spc,&bps,&freeclust,&cluster) ;
    Label3->Caption = CommaAdd(spc);
    Label4->Caption = CommaAdd(bps);
    Label5->Caption = CommaAdd(cluster);

    int free_bytes = freeclust * spc * bps;
    Label6->Caption = CommaAdd (free_bytes);
  }
  else
  {
    ShowMessage ("Problem reading drive information.");
  }
}

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  GetDriveInfo("C:\\");
}
```

После компиляции и запуска этого приложения информация о диске предстанет перед вашими глазами, когда вы щелкнете на кнопке. Если вас интересует не диск `C:`, а какой-то другой, сделайте соответствующую замену спецификатора устройства.

Как видите, ключевыми в этой программе являются функции `GetDiskFreeSpace()` и `GetVolumeInformation()`. Обратите внимание, что информация, содержащаяся в целочисленной переменной `serialnumber`, преобразуется в строку, поскольку функция `GetVolumeInformation()` возвращает значение типа `DWORD`, и поэтому для удобства чтения его лучше преобразовать в строку ASCII.

Создание мигающих уведомлений

В группах новостей часто возникает потребность найти определенное окно или установить текст окна. Для поддержки этой возможности можно использовать такие популярные функции, как `FlashWindowEx()`, `FindWindow()` и `SetWindowText()`. В этом разделе мы и рассмотрим эти функции.

FlashWindowEx()

FlashWindowEx() — НОВАЯ функция Win32 API, поддерживаемая в Windows 98 и Windows 2000. Ее можно использовать вместо не такой уж старой функции FlashWindow(), которая просто создает мигание окна, чтобы уведомить пользователя о каком-либо событии или о том, что окно готово принять на себя фокус. Эти могучие функции (служащие в качестве дополнения к операционной системе) объявлены в заголовочном файле WINUSER.H.

Функция FlashWindow() может просто сделать окно мигающим, но не на постоянной основе, а пока не придет сигнал от таймера. Однако в функции FlashWindowEx() предусмотрен встроенный таймер, и новая функция отличается большей ошибкоустойчивостью. При установке таймера функции FlashWindowEx() можно задать или миллисекунды, или количество миганий. Также можно установить флаги окончания мигания.

Возможно, вы уже заметили использование этой новой функции при работе в среде Windows 98. Этим средством также пользуются такие известные программы, как Instant Messenger (America Online), Yahoo Pager и ICQ.

Объявление этой функции имеет следующий вид.

```
BOOL FlashWindowEx {
  PFLASHWINFO pfwi  // Указатель на структуру с информацией о
                    // статусе миганий.
);
```

Функция FlashWindowEx() принимает указатель на структуру, которая хранит информацию о характере ее выполнения. Возвращаемое значение представляет собой статус окна до обращения к функции FlashWindowEx(). Если окно было активным до вызова, функция возвратит ненулевое значение; в противном случае — нуль. Функция FlashWindowEx() требует указатель на структуру, содержащую информацию о мигании. Эта структура обеспечивает основные ингредиенты, заставляющие функцию FlashWindowEx() работать определенным образом. Функция FlashWindow() более проста в применении, но приведенные ниже примеры убедят вас в простоте и эффективности новой функции FlashWindowEx().

Структура FLASHINFO имеет следующий формат.

```
typedef struct {
  UINT  cbSize;
  HWND  hwnd;
  DWORD dwFlags;
  UINT  uCount;
  DWORD dwTimeout;
}  FLASHWINFO, *PFLASHWINFO;
```

Когда эта структура заполняется и передается как параметр функции FlashWindowEx(), она управляет способом выполнения этой функции. Член dwFlags — это настоящая рабочая лошадка в определении характера мигания. Его значения описаны в табл. 24.22.

Таблица 24.22. Флаги функции FlashWindowEx()

Флаг	Значение
FLASHW_STOP	Останов мигания. Система восстанавливает исходное состояние окна
FLASHW_CAPTION	Мигание заголовка окна
FLASHW_TRAY	Мигание кнопки на панели задач
FLASHW_ALL	Мигание заголовка окна и кнопки на панели задач. Это эквивалентно установке флагов FLASHW_CAPTION и FLASHW_TRAY

Окончание табл. 24.22

Флаг	Значение
FLASHW_TIMER	Непрерывное мигание до тех пор, пока не будет установлен флаг FLASHW_STOP
FLASHW_TIMERNOFG	Непрерывное мигание до тех пор, пока окно не будет переведено на передний план

Из этой таблицы становится ясно, как использовать и управлять функцией FlashWindowEx(), чтобы обеспечить мигание окна.

FindWindow()

Если необходимо найти определенное окно, это можно легко сделать с помощью функции Win32 API FindWindow(). Эта функция найдет окно верхнего уровня на основе его заголовка и имени класса и возвратит его дескриптор. Однако функция FindWindow() не выполняет поиск среди дочерних окон или окно другого уровня. С такой задачей может справиться функция FindWindowEx().

Формат функции FindWindow() имеет следующий вид.

```
HWND FindWindow(
  LPCTSTR lpClassName,  // имя класса
  LPCTSTR lpWindowName  // имя окна
);
```

Параметры, передаваемые функции FindWindow(), — lpClassName и lpWindowName — представляют собой указатели на строки с завершающими null-символами. Параметр lpClassName может принять имя класса окна, а параметр lpWindowName — заголовок окна.

Функция FindWindow() возвращает дескриптор окна, если таковое было найдено.

SetWindowText()

Функция SetWindowText() установит (или изменит) строку заголовка окна, если таковая существует. Если же окно является элементом управления (например, кнопкой), то заменяется лишь текст кнопки. Однако с помощью этой функции нельзя изменить текст элемента управления в другом приложении.

```
BOOL SetWindowText(
  HWND hWnd,         // Дескриптор окна или элемента управления.
  LPCTSTR lpString   // Заголовок или текст.
);
```

Функция SetWindowText() довольно проста в применении. Параметр hWnd — это дескриптор окна, в которое вы собираетесь внести изменения. Параметр lpString — это указатель на строку с завершающим null-символом, содержимое которой заменит существующий текст строки заголовка окна.

А теперь соберем все воедино

В примере, приведенном в листинге 24.11, мы создаем программу, содержащую кнопку. В обработчике события этой кнопки будет послано сообщение редактору Notepad. Если Notepad запущен, будет изменен его заголовок, который затем замигает.

1. Запустите C++Builder.
2. Поместите в форму кнопку.
3. Дважды щелкните на этой кнопке, чтобы создать событие OnClick, и введите код, содержащийся в листинге 24.11.

Листинг 24.11. Использование функции `FlashWindow()` для отправки уведомления пользователю

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

 HWND hHandle = FindWindow (NULL, "Untitled - Notepad");

  FLASHWINFO pf;
  pf.cbSize = sizeof(FLASHWINFO);
  pf.hwnd = hHandle;
  pf.dwFlags = FLASHW_TIMER|FLASHW_ALL;
  pf.uCount = 8;
  pf.dwTimeout  = 75;

   FlashWindowEx(&pf);

   if(hHandle)
     SetWindowText(hHandle, "Notepad");
}
```

4. Запустите программу, нажав клавишу <F9> или щелкнув на кнопке Run (с изображением зеленой стрелки), которая напоминает кнопку видеомагнитофона Play.
5. После запуска программы сверните C++Builder.
6. Запустите редактор Notepad.
7. Сверните Notepad.

Щелкните на кнопке в своей программе, и вы увидите, что заголовок редактора Notepad изменится: с `untitled - Notepad` на `Notepad`. Причем заголовок будет быстро мигать. Теперь нужно привести более подробные разъяснения.

```
HWND hHandle = FindWindow (NULL, "Untitled - Notepad");
```

При выполнении этой строки считывается дескриптор главного окна редактора Notepad. После запуска Notepad и щелчка на кнопке программа обнаруживает редактор Notepad по заголовку его окна и возвращает дескриптор этого окна. Заголовок окна отображается на экране, а дескриптор присваивается переменной `hHandle`.

Например, если вы сохранили текстовый файл под именем `test1`, Notepad будет содержать в заголовке окна имя `test1`. Это имя станет новым именем.

Однако не все окна одинаковы. Например, если бы вы запустили другое приложение, то оно имело бы совсем другое имя.

Как же быть в таком случае? Очень просто! Если вы во время разработки знаете имя окна другого приложения, можете реализовать идею нашей программы. Если вы не знаете имя окна, просто используйте описанные выше функции-контейнеры для получения дескриптора окна или опросите все окна и поместите их имена в список.

Теперь рассмотрим структуру, требуемую функцией `FlashWindowEx()`.

```
FLASHWINFO pf;
  pf.cbSize = sizeof(FLASHWINFO);
  pf.hwnd = hHandle;
  pf.dwFlags = FLASHW_TIMER|FLASHW_ALL;
  pf.uCount = 8;
```

```
pf.dwTimeout  = 75;

FlashWindowEx(&pf);
```

Структура FLASHWINFO содержит всю информацию, которая нужна функции FlashWindowEx() для выполнения ее задачи. Здесь объявлена новая переменная pf типа этой структуры. Затем она инициализируется и передается в качестве параметра функции FlashWindowEx(). Теперь рассмотрим члены этой структуры.

Член cbSize используется для идентификации размера структуры FLASHWINFO.

Член hwnd обеспечивает дескриптор окна приемника. В предыдущем примере дескриптор окна редактора Notepad был получен следующим образом.

```
HWND hHandle = FindWindow (NULL, "Untitled - Notepad");
```

Член dwFlags представляет собой флаги для функции FlashWindowEx(). Если использовать флаг FLASHW_TRAY, то будет мигать только кнопка на панели задач. В нашем примере заказывается мигание кнопки и заголовка. Флаг FLASHW_TIMER предусматривает установку флага FLASHW_STOP.

Параметр uCount содержит желаемое количество миганий окна. В нашем примере было заказано восемь раз. Функция FlashWindowEx() "знает", что после восьмого мигания нужно вернуться и установить флаг FLASHW_STOP.

Параметр dwTimeout содержит количество миллисекунд, в течение которых окно должно мигать. В нашем примере мигание завершается довольно быстро, но вы вольны изменить установки и понаблюдать за изменениями в поведении программы.

Как упоминалось выше, функция FlashWindowEx() может уведомить пользователя о возникновении ошибки или некоторого события, не прибегая к услугам окон сообщений. Процедура уведомления позволяет сообщить пользователю, что окно приложения не имеет фокуса. Это особенно удобно, если вы не хотите, чтобы работа программы пользователя прерывалась окнами сообщений. Кроме того, для закрытия окна сообщения требуются дополнительные действия пользователя. Вместо отображения окна сообщения, функция FlashWindowEx() уведомит пользователя миганием.

Добавление системной поддержки

Иногда необходимо усилить приложение системной поддержкой, например наделить его способностью быстро заблокировать рабочую станцию NT, запретить комбинацию клавиш <Ctrl+Alt+Delete> или завершить работу системы (либо перезагрузить ее). В арсенале интерфейса Win32 API есть немало функций, способных облегчить выполнение этой задачи. Однако очень важно знать об опасностях, подстерегающих вас при реализации этих возможностей. Если вы (несмотря на предупреждение) все-таки собираетесь реализовать системную поддержку, как можно чаще сохраняйте свою работу, в противном случае вы можете пожалеть, что не выбрали в свое время другую профессию.

Блокировка рабочих станций NT

Самое время познакомить вас с еще одной новой функцией Win32 API — LockWorkStation(), которая используется для автоматической блокировки системы NT. Функция LockWorkStation() не принимает параметров, поэтому очень просто заблокировать рабочие станции и серверы. Применение этой функции аналогично применению хорошо известной комбинации клавиш <Ctrl+Alt+Delete> и команды Lock Workstation (Заблокировать рабочую станцию). После этого система остается заблокированной до тех пор, пока ее не разблокируют. Эта функция объявлена в заголовочном файле WINUSER.H.

Выполнив следующую строку программы, можно автоматизировать блокировку системы, которая обычно выполняется вручную в среде NT. К сожалению, эта функция не поддерживается в Windows 9x.

```
LockWorkStation();
```

Завершение работы системы

Начинающие программисты часто спрашивают, как автоматизировать завершение работы Windows PC. Это можно сделать с помощью функций `ExitWindowsEx()` или `ExitWindows()`. Следующая строка программы демонстрирует использование API-функции `ExitWindowsEx()`.

```
ExitWindowsEx(EWX_SHUTDOWN, 0);
```

Эта функция не такая новая, как `LockWorkStation()`, но возможность выключить систему — сильный рычаг в механизме управления ею. Если ваш компьютер оснащен средствами экономичного расходования электроэнергии в BIOS, функция `ExitWindowEx()` автоматически отключит и систему питания вашего компьютера.

Использование функции `ExitWindowsEx()` — простой способ завершить работу системы Windows, но в ваших силах сделать гораздо больше, чем просто остановить Windows. Функция `ExitWindowEx()`, помимо флага `EWX_SHUTDOWN`, переданного в качестве параметра в приведенном выше примере, может использовать и другие флаги. Эта функция имеет следующий формат.

```
BOOL ExitWindowsEx(
    UINT uFlags,          // Операция завершения работы системы.
    DWORD dwReserved      // Зарезервированный параметр.
   );
```

Параметр `uFlags` должен содержать флаг, определяющий, как будет закрыта система. Описание всех флагов приведено в табл. 24.23.

Таблица 24.23. Флаги функции завершения работы системы

Флаг	Значение
`EWX_FORCE`	Завершает процессы. При установке этого флага Windows не отправляет сообщения `WM_QUERYENDSESSION` и `WM_ENDSESSION` приложениям, работающим в данный момент в системе. Это может привести к потере данных у приложений, поэтому этот флаг следует использовать в аварийных ситуациях
`EWX_LOGOFF`	Завершает все процессы, работающие в контексте безопасности процесса, который вызвал функцию `ExitWindowsEx()`. Затем завершается работа пользователя в системе
`EWX_POWEROFF`	Завершает работу системы и отключает питание. Система должна поддерживать средство отключения питания. Windows NT: вызывающий процесс должен обладать привилегией `SE_SHUTDOWN_NAME`. Windows 95: привилегии безопасности не поддерживаются или не требуются
`EWX_REBOOT`	Завершает работу, а затем перезагружает систему. Windows NT: вызывающий процесс должен обладать привилегией `SE_SHUTDOWN_NAME`
`EWX_SHUTDOWN`	Доводит работу системы до момента, когда безопасно отключить питание. Все файловые буферы переписываются на диск, а все работающие процессы останавливаются. Windows NT: вызывающий процесс должен обладать привилегией `SE_SHUTDOWN_NAME`

Параметр `dwReserved` пока не используется.

Анимационные эффекты

Новый элемент управления оконного типа, введенный в Windows 98 и Windows 2000, позволяет создавать анимационные эффекты, которые приятно удивят ваших пользователей. Новая функция API `AnimateWindows()` служит для отображения анимационных процессов в формах. Работая с Windows 98, вы, вероятно, уже могли увидеть подобные эффекты. Прогнозируется, что вскоре все больше программистов будут использовать для окон эффекты подобного типа. Эта новая API-функция имеет несколько параметров, корректно управляя которыми, можно получить очень интересные результаты. При надлежащем программировании можно создать потрясающие эффекты как для отдельного компонента, так и для всей формы.

`AnimateWindow()`

Формат функции `AnimateWindow()` кажется очень простым, но если не использовать его правильно, ничего хорошего не получится. Итак, эта функция принимает следующие параметры.

```
BOOL AnimateWindow(
  HWND hwnd,     // дескриптор окна
  DWORD dwTime,  // продолжительность анимации
  DWORD dwFlags  // тип анимации
);
```

Параметр `hwnd` — дескриптор окна, где вы собираетесь применить анимацию.

Параметр `dwTime` указывает время в миллисекундах, в течение которого будет выполняться анимация.

Параметр `dwFlags` представляет различные флаги и типы анимации, которые можно выполнить (табл. 24.24).

Таблица 24.24. Флаги функции `AnimateWindow()`

Флаг	Значение
`AW_SLIDE`	Использует слайдовую анимацию. По умолчанию используется катушечная. Этот флаг игнорируется при использовании флага `AW_CENTER`
`AW_ACTIVATE`	Активизирует окно. Не используйте это значение совместно с флагом `AW_HIDE`
`AW_BLEND`	Использует эффект плавного замирания (увядания). Этот флаг используется только в том случае, когда дескриптор `hwnd` относится к окну верхнего уровня
`AW_HIDE`	Скрывает окно. По умолчанию окно отображается
`AW_CENTER`	Создает впечатление сворачивания вовнутрь, если используется флаг `AW_HIDE`. Если флаг `AW_HIDE` не используется, кажется, что окно раскрывается наружу
`AW_HOR_POSITIVE`	Анимация окна слева направо. Этот флаг можно использовать с помощью слайдовой или катушечной анимации. Он игнорируется при использовании с флагами `AW_CENTER` или `AW_BLEND`
`AW_HOR_NEGATIVE`	Анимация окна справа налево. Этот флаг можно использовать с помощью слайдовой или катушечной анимации. Он игнорируется при использовании с флагами `AW_CENTER` или `AW_BLEND`
`AW_VER_POSITIVE`	Анимация окна сверху вниз. Этот флаг можно использовать с помощью слайдовой или катушечной анимации. Он игнорируется при использовании с флагами `AW_CENTER` или `AW_BLEND`

Окончание табл. 24.24

Флаг	Значение
AW_VER_NEGATIVE	Анимация окна снизу вверх. Этот флаг можно использовать с помощью слайдовой или катушечной анимации. Он игнорируется при использовании с флагами AW_CENTER или AW_BLEND

Функция может возвращать либо нуль, либо положительное значение. Функция терпит неудачу, если в заданном окне делается попытка отобразить уже видимое или скрыть уже скрытое окно.

Процедурам, работающим с окном и его дочерними окнами, может понадобиться обработка сообщений WM_PRINT или WM_PRINTCLIENT. Диалоговые окна и элементы управления уже обрабатывают сообщение WM_PRINTCLIENT. Стандартная оконная процедура уже обрабатывает сообщение WM_PRINT.

Теперь хотелось бы опробовать эту функцию.

1. Запустите C++Builder и создайте новое приложение.
2. В обработчики событий формы FormShow и FormClose введите следующий код.

```
void __fastcall TForm1::FormShow(TObject *Sender)
{
  AnimateWindow(Form1->Handle, 2000, AW_BLEND | AW_HOR_POSITIVE);
}
//---------------------------------------------------------------

void __fastcall TForm1::FormClose(TObject *Sender, TCloseAction &Action)
{
  AnimateWindow(Form1->Handle, 1000,
                          AW_HIDE | AW_BLEND | AW_VER_POSITIVE);
}
```

3. Скомпилируйте программу, запустите ее и наслаждайтесь увиденным.

Обратите внимание на то, как медленно (в течение двух секунд) появляется форма приложения, а также на ее "увядание" при выходе из программы.

Вместо мгновенного "выталкивания" приложения на экран можно добиться некоторой "степенности".

На заметку Функция AnimateWindow() может не работать для всех пользователей, поскольку средство Animate Windows (Анимация окон) может быть запрещено на панели управления Windows.

Использование флага AW_HIDE в событии FormClose предписывает функции AnimateWindow() скрыть приложение.

Хотите что-нибудь получше? Замените событие FormClose следующим кодом.

```
AnimateWindow(Form1->Handle, 1000,
        AW_HIDE | AW_SLIDE | AW_HOR_POSITIVE | AW_VER_NEGATIVE);
```

При выполнении этой строки кода создается эффект анимации окна по диагонали. Чтобы добиться анимации окна в различных диагональных позициях, можно скомбинировать флаг AW_HOR_POSITIVE или AW_HOR_NEGATIVE с флагом AW_VER_POSITIVE или AW_VER_NEGATIVE.

Как вам теперь? Замените тот же код в событии FormClose следующим.

```
AnimateWindow(Form1->Handle, 1000, AW_HIDE | AW_SLIDE | AW_VER_POSITIVE);
```

Такое сочетание флагов заставляет окно “оседать”, а затем исчезнуть.

“Поиграйтесь” и с параметрами времени. Вместо 1 000 миллисекунд, попробуйте значение 2 500. А если вы при этом используйте флаг AW_BLEND, то форма будет напоминать прокрутку титров кино.

Как видите, компания Microsoft не пожалела времени на интерфейс GUI и внесла такое разнообразие в процесс создания окон, что у пользователя может захватить дух.

Придание формы приложениям

Системное программирование — довольно забавная штука. Манипулирование формами и придание им “крутого” вида и поведения могут произвести на пользователя потрясающее впечатление. В этом разделе мы рассмотрим идею областей и ряд таких родственных функций, как CombinRgn(), CreateEllipticRgn() и GetClientRect(). Эти функции способны управлять внешним видом окна. Вместо унылого прямоугольника можно реализовать другое решение, используя области окна. C++Builder чрезвычайно упрощает сочетание функций интерфейса Win32 API для создания областей в системе. Упомянутые (и другие) функции API объявлены в заголовочном файле WINGDI.H.

Что же такое область (region)? Область — это площадь окна. Все что, находится вне области, не считается частью окна. Окно отрезает все, находящееся вне области, что придает ему другой вид.

CreateRoundRectRgn()

Начнем, пожалуй, с функции CreateRoundRectRgn(). Эта функция создает прямоугольное окно со скругленными углами.

Функция CreateRoundRectRgn() принимает следующие параметры.

```
HRGN CreateRoundRectRgn(

  int nLeftRect,    // X-координата верхнего левого угла области.
  int nTopRect,     // Y-координата верхнего левого угла области.
  int nRightRect,   // X-координата нижнего правого угла области.
  int nBottomRect, // Y-координата нижнего правого угла области.
  int nWidthEllipse,   // Высота эллипса, используемого для скругления углов.
  int nHeightEllipse   // Ширина эллипса, используемого для скругления углов.
   );
```

Параметр nLeftRect задает координату X верхнего левого угла области.

Параметр nTopRect задает координату Y верхнего левого угла области.

Параметр nRightRect задает координату X нижнего правого угла области.

Параметр nBottomRect задает координату Y нижнего правого угла области.

Параметр nWidthEllipse задает ширину эллипса, используемого для создания скругленных углов прямоугольника.

Параметр nHeightEllipse задает высоту эллипса, используемого для создания скругленных углов прямоугольника.

В следующем примере показано, как создать окно со скругленными углами.

```
RECT R  = GetClientRect();
 HRGN MyRegion = CreateRoundRectRgn(0,0,150,110,15,10);
 SetWindowRgn(Handle , MyRegion , true);
```

Все очень просто. Самое трудное — определить координаты. Часто приходится немного “поиграть” с ними, пока добьешься желаемого вида, но это — единственный недостаток.

Итак, не мудрствуя лукаво, создадим программу, которая будет иметь полосы прокрутки (чтобы мы могли увидеть все-все), отображать координаты и в динамике менять вид нашего окна.

Запустите C++Builder. Поместите в форму компоненты, перечисленные в табл. 24.25.

Таблица 24.25. Компоненты, помещаемые в форму

Компонент	Свойство
Form1	Width = 369 Height = 416
Label1	Left = 16 Top = 53
Label2	Left = 16 Top = 120
Label3	Left = 16 Top = 37 Caption = значению X-координаты нижнего правого угла области
Label4	Left = 16 Top = 104 Caption = значению Y-координаты нижнего правого угла области
Label5	Left = 16 Top = 176 Caption = значению X-координаты верхнего левого угла области
Label6	Left = 16 Top = 192
Label7	Left = 16 Top = 248 Caption = значению Y-координаты верхнего левого угла области
Label8	Left = 16 Top = 264
Label9	Left = 16 Top = 312 Caption = высоте эллипса, используемого для скругления углов
Label10	Left = 16 Top = 312
Label11	Left = 16 Top = 368 Caption = ширине эллипса, используемого для скругления углов
Label12	Left = 16 Top = 368
TrackBar	Name = Track1 Left = 8 Top = 144 Width = 297 Height = 27 Frequency = 1

Окончание табл. 24.25

Компонент	Свойство
TrackBar	Left = 8 Top = 216 Width = 305 Height = 27
TrackBar	Name = Track3 Left = 8 Top = 8 Width = 297 Height = 27 Max = 230 Min = 70 Frequency = 5 Position = 190
TrackBar	Name = Track4 Left = 8 Top = 72 Width = 297 Height = 27 Max = 100 Frequency = 5
TrackBar	Name = Track5 Left = 8 Top = 280 Width = 233 Height = 27
TrackBar	Name = Track6 Left = 8 Top = 336 Width = 233 Height = 27

Вам останется лишь поместить нужный код в исходный файл формы и в обработчик события `OnChange` компонента `TrackBar3`. Компонент `Trackbar` находится во вкладке Win32 палитры компонентов. К событию `OnChange` компонента `TrackBar3` можно получить доступ с помощью окна Object Inspector.

Этот код представлен в листинге 24.12.

Листинг 24.12. Создание нестандартной формы с помощью функции `CreateRoundRectRgn()`

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

#include "Unit1.h"
//---------------------------------------------------------------
#pragma resource "*.dfm"
```

```
TForm1 *Form1;
//---------------------------------------------------------------
__fastcall TForm1::TForm1(TComponent* Owner)
    : TForm(Owner)
{
   Form1->Height = 416;
   Form1->Width  = 369;

  Track3->Max = Form1->Width+10;
  Track3->Min = 70;
  Track3->Position = Form1->Width;  // Начнем с полного размера формы.

  Track4->Max = Form1->Height+10;
  Track4->Min = 195;
  Track4->Position = Form1->Height;  // Начнем с полного размера формы.

  Track1->Max = 20; // Не разрешается заходить за позицию на полосе.
  Track1->Min = 0;
  Track1->Position = 0;

  Track2->Max = 230;
  Track2->Min = 0;
  Track2->Position = 0;

  Track5->Max = 500;
  Track5->Min = 0;
  Track5->Position = 31;

  Track6->Max = 500;
  Track6->Min = 0;
  Track6->Position = 31;
}
//---------------------------------------------------------------
void __fastcall TForm1::Track3Change(TObject *Sender)
{
 // Создаем дескриптор области и получаем дескриптор от функции
 // CreateRoundRgn(), если это допустимо.

 // Позиции компонентов Trackbar необходимо задать в качестве
 // параметров функции.
 HRGN MyRegion = CreateRoundRectRgn(Track1->Position,
                             Track2->Position, Track3->Position,
          Track4->Position, Track5->Position, Track6->Position);

 // Устанавливаем область окна с помощью этой функции.
 // Параметры: SetWindowRgn(Дескриптор окна, Дескриптор области HRGN,
 //                         Режим перерисовки после установки области)
 SetWindowRgn(Handle , MyRegion , true);

 // Надписи будут отслеживать все позиции.
```

```
 Label1->Caption = Track3->Position;
 Label2->Caption = Track4->Position;
 Label6->Caption = Track1->Position;
 Label8->Caption = Track2->Position;
 Label10->Caption = Track5->Position;
 Label12->Caption = Track6->Position;
 Form1->Update ();
}
```

Не забудьте установить все остальные события `OnChange` компонентов `TrackBar` по аналогии с компонентом `TrackBar3`. Скомпилируйте и запустите программу.

В данном примере компоненты `TrackBar` используются для изменения позиции окна. Область окна модифицируется с помощью функции `CreateRoundRectRgn()`. Надписи встроены в заголовок формы и отображают позиции области окна. При использовании функций Win32 API вы можете опираться на этот пример, чтобы видеть координаты, а не угадывать их.

`CreateEllipticRgn()`

А что, если мы захотим, чтобы окна выглядели совсем по-другому? Существует еще одна функция, `CreateEllipticRgn()`, которая придает окну эллипсовидную форму. Вообще-то, функция сделает ваше окно круглым или овальным — в зависимости от переданных ей параметров. Поэтому сразу же приступаем к их рассмотрению.

```
HRGN CreateEllipticRgn(

    int nLeftRect,
                    // X-координата верхнего левого угла
                    //   ограничительного прямоугольника.
    int nTopRect,
                    // Y-координата верхнего левого угла
                    //   ограничительного прямоугольника.
    int nRightRect,
                    // X-координата нижнего правого угла
                    //   ограничительного прямоугольника.
    int nBottomRect
                    // Y-координата нижнего правого угла
                    //   ограничительного прямоугольника.
 );
```

Параметр `nLeftRect` задает координату X верхнего левого угла ограничительного прямоугольника эллипса.

Параметр `nTopRect` задает координату Y верхнего левого угла ограничительного прямоугольника эллипса.

Параметр `nRightRect` задает координату X нижнего правого угла ограничительного прямоугольника эллипса.

Параметр `nBottomRect` задает координату Y нижнего правого угла ограничительного прямоугольника эллипса.

Эти параметры во многом похожи на параметры функции `CreateRoundRectRgn()`. В следующем примере показано использование функции `CreateEllipticRgn()`.

```
HRGN HRegion
HRegion = CreateEllipticRgn(0, 0, Form1->Width, Form1->Height);
SetWindowRgn(Handle , hRgn1 , true);
```

При выполнении этого фрагмента программы ваша форма примет вид овала. Мы указали начальную позицию `Left(0) Top(0)` и задали значения ширины и высоты окна, получив в результате окно в виде овала.

CombineRgn()

Функция `CombineRgn()` выделяется среди прочих своей мощью. Она может принять заданные вами области и объединить их. У программ сегодняшнего дня, придающих своим окнам нестандартную форму, функция `CombineRgn()` пользуется большой популярностью. Сочетая функции `CreateEllipticRgn()`, `CreateRoundRect()` и `CreatePolygonRgn()`, можно сделать окна очень эффектными.

Функция `CombineRgn()` принимает следующие параметры.

```
int CombineRgn(
    HRGN hrgnDest,        // Дескриптор области приемника.
    HRGN hrgn1,           // Дескриптор области источника.
    HRGN hrgn2,           // Дескриптор области источника.
    int fnCombineMode     // Режим объединения областей.
   );
```

Параметр `hrgnDest` идентифицирует новую область, размеры которой определяются комбинацией двух других областей. (Эта область должна существовать до вызова функции `CombineRgn()`.)

Параметр `hrgn1` идентифицирует первую из двух объединяемых областей.

Параметр `hrgn2` идентифицирует вторую из двух объединяемых областей.

Параметр `fnCombineMode` задает режим, определяющий, как именно будут объединены две области. Этот параметр может принимать одно из значений, описанных в табл. 24.26.

Таблица 24.26. Флаги функции `CombineRgn()`

Флаг	Значение
`RGN_AND`	Создает пересечение двух исходных областей
`RGN_COPY`	Создает копию области, идентифицируемой параметром `hrgn1`
`RGN_DIFF`	Объединяет части области `hrgn1`, которые не являются частями области `hrgn2`
`RGN_OR`	Объединяет две исходные области
`RGN_XOR`	Объединяет две исходные области, за исключением перекрывающихся частей

Возвращаемое значение определяет тип полученной в результате области, т.е. “на выходе” функции `CombineRgn()` можно получить одно из значений, перечисленных в табл. 24.27.

Таблица 24.27. Значения, возвращаемые функцией `CombineRgn()`

Возвращаемое значение	Описание
`NULLREGION`	Область пуста
`SIMPLEREGION`	Областью является один прямоугольник
`COMPLEXREGION`	Область состоит не только из одного прямоугольника
`ERROR`	Область не создана

Интересно отметить, что указываемые области совсем не обязательно должны быть отдельными. Например, если в качестве параметров передать такие три области, как Region1, Region1 и Region2, нетрудно заметить, что исходная область Region1 должна содержать и результирующую область. В этом нет ничего страшного. Но не следует забывать, что область приемника должна быть объявлена до вызова API-функции.

А вот и пример использования функции CombineRgn().

```
HRGN Region1, Region2;
    Region1 = CreateRectRgn(0, 0, 100, 100);
    Region2 = CreateRectRgn(50, 50, 150, 150);
    CombineRgn(Region1, Region1, Region2, RGN_XOR);
    SetWindowRgn(Handle , hRgn1 , true);
```

Выполнив этот пример, вы увидите свою форму в виде двух прямоугольников с прямоугольной "дырой" посередине. "Дыра" образовалась в результате использования флага RGN_XOR. Две области перекрываются, но объединяются без пересекающихся областей. Если бы использовался флаг RGN_OR, объединение произошло бы с участием пересечения этих областей.

Кроме того, мы использовали область Region1 в качестве источника и приемника. Спросите, зачем? А затем, что необходимо задать область приемника, и нас вполне устраивает, что Region1 будет перезаписана объединением областей Region1 и Region2. После передачи параметров функции становятся известны координаты, которые используются для области приемника Region1.

Однако не будем останавливаться на полпути. Выше уже упоминалась функция CreatePolygonRgn(). Ей стоит уделить внимание.

Функция CreatePolygonRgn() принимает следующие параметры.

```
HRGN CreatePolygonRgn(

    CONST POINT *lppt,     // Указатель на массив точек.
    int cPoints,           // Количество точек в массиве.
    int fnPolyFillMode     // Режим построения многоугольника.
);
```

Параметр lppt указывает на массив структур POINT, которые определяют вершины многоугольника. Предполагается, что многоульник замкнутый. Каждая вершина должна задаваться только однажды.

Параметр CPoints задает количество точек в массиве.

Параметр fnPolyFillMode задает режим заполнения области. Этот параметр может иметь значения, перечисленные в табл. 24.28.

Таблица 24.28. Возможные режимы, задаваемые параметром fnPolyFillMode

Параметр	Значение
ALTERNATE	Выбирает режим чередования (на каждой сканирующей строке заполняет область между нечетными и четными сторонами многоугольника)
WINDING	Выбирает режим изгиба (заполняет любую область ненулевым значением изгиба)

Чтобы передать соответствующие параметры функции CreatePolygonRgn(), можно использовать как интерфейс Win32 API, так и структуру C++Builder POINT. Эта функция при

создании области опирается на пиксельные позиции точек, и этот процесс напоминает соединение точек.

Структура POINT определяет координаты X и Y объекта. Ее просто нужно заполнить. Для задания координат нескольких точек достаточно объявить переменную структуры POINT с использованием массива.

Не пора ли перейти к примеру? Ниже приводится пример использования функции CreatePolygonRgn(). Прошу меня простить, что я остановил свой выбор на создании знака "Стоп", но он прекрасно подходит для примера. Здесь мы воспользуемся еще одной могучей функцией Win32 API — GetClientRect(). Эта функция получает координаты прямоугольника клиента и возвращает значения для структуры RECT. Структура RECT содержит верхние левые и нижние правые координаты объекта. Функция GetClientRect() является частью библиотеки VCL C++Builder.

1. Запустите C++Builder и создайте новое приложение.
2. На главной форме приложения, Form1, установите цвет равным clRed и поместите в форму кнопку.
3. Вставьте следующий код в обработчик события OnClick.

```
void __fastcall TForm1::BitBtn1Click(TObject *Sender)
{
 Close();
}
```

Затем в модуль исходного кода формы вставьте следующий код.

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

 RECT R;
   R = GetClientRect();
   POINT p[8];
   p[0].x =87; p[0].y =0;
   p[1].x =25; p[1].y =59;
   p[2].x =24; p[2].y =123;
   p[3].x =79; p[3].y =176;
   p[4].x =171; p[4].y =173;
   p[5].x =207; p[5].y =123;
   p[6].x =208; p[6].y =59;
   p[7].x =156; p[7].y =0;

   HRGN MyRegion;

   MyRegion =  CreatePolygonRgn(p, 8, ALTERNATE);

   SetWindowRgn(Handle, MyRegion, true);
}
```

В пункте 3 мы объявили структурированную переменную RECT, которой присвоили координаты формы клиента, возвращенные функцией GetClientRect().

Затем мы объявили переменную p типа POINT, которая представляет собой массив точек, предназначенный для создания знака "Стоп" (рис. 24.6).

Объявив все массивы точек, мы создали дескриптор `MyRegion`. Для этого была вызвана функция `CreatePolygonRgn()`, которая (в случае успешного выполнения) вернет дескриптор области `MyRegion`.

Обратите внимание на параметры функции `CreatePolygonRgn()`. Мы передали ей структурированную переменную `p` типа `POINT`, т.е. функция в качестве первого параметра принимает адрес структуры. Следом мы должны передать количество точек в структуре. Если бы мы передали число 4, то получили бы довольно забавную форму. Наконец, в качестве режима заполнения формы передано значение `ALTERNATE`.

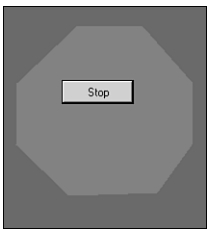


Рис. 24.6. Знак "Стоп"

После этого мы изменяем внешний вид окна, вызвав функцию `SetWindowRgn()`, которая изменяет область окна.

После запуска приложения вы должны увидеть "Стоп"-знак красного цвета с кнопкой в форме, ожидающей закрытия.

Мы рассмотрели лишь основы применения областей, чтобы вы поняли, как работать с ними в среде C++Builder. Чтобы больше узнать о том, как создавать окна с помощью областей, посетите Web-узел Microsoft по адресу: `http://www.microsoft.com/msdn`. В качестве ключевых слов при поиске нужной информации используйте `window regions` (области окон) или `irregular windows` (нестандартные окна).

Написание аплетов панели управления в стиле ретро

Аплеты панели управления (Control Panel) представляют собой небольшие программы, которые могут конфигурировать среду Windows, других приложений или какого-либо оборудования. Аплеты панели управления, больше известные просто как *аплеты*, на самом деле являются библиотеками динамической компоновки (DLL). Однако эти специфические DLL переименовываются и получают расширение `.cpl`, которое означает не что иное как *Control Panel applet* и позволяет пользователю или разработчику отличать их от обычных DLL.

Этот раздел, как нетрудно догадаться, посвящен аплетам панели управления. Мы рассмотрим основы их создания, напишем маленькую утилиту и превратим ее в аплет панели управления. C++Builder 5 уже содержит мастер аплетов панели управления (Control Panel Applet Wizard), с помощью которого можно быстро написать аплет, но мы покажем вам другой способ. Этот способ (причем вполне простой) заключается в использовании интерфейса Win32 API, который в сочетании с сервисом C++Builder позволяет быстро добиться успеха.

Понятие об аплетах панели управления

Главная цель аплета панели управления — разрешить пользователю взаимодействие с конфигурацией системы или приложения. Аплет реагирует на двойной щелчок мышью в пределах панели управления, активизируя ее. Обычно аплет панели управления используется для конфигурирования среды больших программ или даже самой (!) Windows. Например, вы написали большую программу, но вместо того, чтобы заставлять пользователя ожидать, пока она загрузится, чтобы сконфигурировать ее, вы могли бы создать утилиту для конфигурирования этой программы и сохранить параметры в системном реестре. Аплеты панели управления можно узнать по пиктограммам и маленьким надписям под ними. Они также содержат дополнительный текст в строке состояния панели управления, кратко описывающий их назначение.

Аплеты панели управления совсем не трудно написать, они только с виду кажутся неприступными. Поэтому давайте-ка сначала подсмотрим, как они работают.

Аплеты панели управления компилируются как специальные библиотеки динамической компоновки (Dynamic Linked Library) и экспортируют стандартную входную функ-

цию CPLApplet(). Эта функция выполняет запросы в форме сообщений панели управления и такие затребованные задачи, как инициализация приложения, получение пиктограмм, отображение внутренних диалоговых окон и управление ими, а также просто закрывает процессы.

При загрузке панель управления всегда принимает различную информацию о каждом аплете. Панель управления — это управляющее приложение, которое руководит каждым аспектом "своих" аплетов. Конечно же, вы могли бы написать свою собственную программу, которая бы посылала сообщения и обрабатывала возвращаемую информацию, как это делает панель управления.

Как только вы выберете команду Start⇨Settings⇨Control Panel (Пуск⇨Настройка⇨Панель управления), панель управления прочитает все аплеты (или файлы с расширением .CPL) из папки \Windows\System или \Winnt\System32. Затем опросит пиктограммы, вычислит количество диалоговых окон в каждом аплете и соберет другую важную информацию. Вот поэтому-то загрузка панели управления и занимает две-три секунды.

Когда загружается панель управления, т.е. управляющая программа, которая вызывает аплеты панели управления, адрес функции CPLApplet() используется для передачи сообщений аплетов. Формат функции CPLApplet() такой.

```
LONG APIENTRY CPlApplet(
    HWND hwndCPl,    // Дескриптор окна панели управления.
    UINT uMsg,       // Сообщение.
    LONG lParam1,    // Первый параметр сообщения.
    LONG lParam2     // Второй параметр сообщения.
   );
```

Параметр hwndCPl идентифицирует главное окно управляющего приложения.

Параметр uMsg задает сообщение, посылаемое приложению панели управления.

Параметр lParam1 задает дополнительную информацию, относящуюся к сообщению.

Параметр lParam2 также задает дополнительную информацию, относящуюся к сообщению.

Значение, возвращаемое функцией CPLApplet(), зависит от конкретного сообщения. Чуть ниже мы рассмотрим это. Обратите внимание, что параметр hwndCPl для диалоговых или других окон требует дескриптор родительского окна.

Когда загружается панель управления или управляющее приложение, функция CPLApplet() вызывается много раз. Возвращаемые значения зависят от переданных параметров и особенно — от параметра uMsg. Аплет необходимо писать строго определенным способом, поскольку сообщения, посылаемые функции CPLApplet(), имеют определенный порядок.

В табл. 24.29 показан порядок сообщений и значений.

Таблица 24.29. Порядок сообщений

Сообщение	Порядок	Операция
CPL_INIT (#define константное значение = 1)	Первый вызов. Немедленный вызов после загрузки .CPL-файла, содержащего аплет	Посылается, чтобы уведомить, что найдена функция CPLApplet(). Параметры lParam1 и lParam2 не используются. Возвращает значение TRUE или FALSE, означающие возможность продолжения работы панели управления. В случае неудачи прекращается управление аплетом и разрываются все связи с .CPL-файлами

Окончание табл. 24.29

Сообщение	Порядок	Операция
CPL_GETCOUNT (#define константное значение = 2)	Вызывается после CPL_INIT. Возвращает ненулевое значение	Посылается для определения количества отображаемых аплетов. Возвращает ненулевое значение. Возвращает количество аплетов, которые нужно отобразить в окне панели управления. Параметры *lParam1* и *lParam2* еще не используются
CPL_INQUIRE (#define константное значение = 3)	Вызывается после CPL_GETCOUNT. Возвращаемое значение больше или равно 1. Функция CPLApplet() будет вызвана один раз для каждого диалогового окна, задавая, какое диалоговое окно использует индексное значение с нулевой базой. Это значение помещается в параметр *lParam1*	Посылается информация о каждом аплете. Функция CPLApplet() предоставляет информацию в диалоговое окно. Параметр *lParam1* указывает на загружаемое диалоговое окно. Параметр *lParam2* указывает на структуру CPLINFO, из которой функция CPLApplet() получает информацию об имени, пиктограмме и пр.
CPL_ NEWINQUIRE (#define константное значение = 8)	Если сообщение CPL_INQUIRE не используется, можно использовать это сообщение. У этих сообщений одинаковы порядок и действия	Выполняются те же операции, что и при отправке сообщения CPL_ENQUIRE, но только *lParam2* указывает на структуру NEWCPLINFO, означающую новые данные для самого .CPL-файла. С целью увеличения производительности в Win 95/NT используется сообщение CPL_ENQUIRE
CPL_DBLCLK (#define константное значение = 5)	Вызывается после того, как пользователь дважды щелкнет на пиктограмме аплета	Посылается после щелчка на пиктограмме аплета. *lParam1* — номер выбранного аплета. *LParam2* — значение lData аплета. Это сообщение должно инициировать диалоговое окно аплета. В действительности это выражается в отображении формы, разработанной для аплета
CPL_STOP (#define константное значение = 6)	Вызывается единожды для каждого диалогового окна сразу перед закрытием приложения. Оно задается параметром *lParam1* с использованием нулевой базы	Функция CPLApplet() должна попытаться освободить любые ресурсы, выделенные для диалогового окна. Посылается для каждого аплета, когда панель управления существует. Параметр *lParam1* — номер аплета. *LParam2* — индексное значение lData аплета. Сделайте здесь всю очистку, связанную с аплетом
CPL_EXIT (#define константное значение = 7)	Посылается один раз. Это сообщение посылается после того, как будет послано последнее сообщение CPL_STOP из индексированных диалоговых окон. Вызывается непосредственно перед тем, как управляющее приложение использует функцию FreeLibrary() для освобождения .CPL-файла, содержащего аплет	Посылается как раз перед тем, как панель управления вызовет функцию FreeLibrary(). Параметры lParam1 и lParam2 не используются. Здесь выполняется очистка, не связанная с аплетом: освобождение ресурсов, памяти и пр.

Работая в соответствии с заданным порядком, функция CPLApplet() при загрузке аплета получает сообщение CPL_INIT с помощью управляющего приложения (панели управления). Функция инициализирует переменные, выделяет память и должна (в случае успеха) возвратить ненулевое значение. При неудачном выполнении функция CPLApplet() возвратит нуль (или значение FALSE) и велит управляющему приложению прекратить "всякие отношения" с аплетом панели управления.

Затем, если сообщение CPL_INIT сработало успешно, управляющее приложение посылает сообщение CPL_GETCOUNT. В этом случае функция CPLApplet() возвращает количество диалоговых окон, которые поддерживаются аплетом панели управления. Например, если у вас предусмотрена поддержка трех диалоговых окон, функция CPLApplet() возвратит число 3.

Затем функция CPLApplet() получает сообщение CPL_INQUIRE и по одному сообщению CPL_NEWINQUIRE для каждого диалогового окна, которое поддерживается аплетом панели управления. Функция заполняет структуру CPLINFO или NEWCPLINFO, содержащую информацию о каждом поддерживаемом диалоговом окне. Эта информация состоит из имени, пиктограммы и описания аплета. Заполните такие поля структуры CPLINFO, как idIcon, idName, idInfo и lData идентификационным номером отображаемой пиктограммы, строковыми значениями ID имени и описания, а также элементом данных типа long, связанным с аплетом #lParam1. Эта информация может быть перехвачена вызывающей процедурой во время выполнения и между сеансами работы.

Обычно в средах Windows 95 и NT (чтобы избежать потерь в производительности) следует использовать сообщение CPL_INQUIRE и игнорировать сообщение CPL_NEWINQUIRE. Сообщение CPL_INQUIRE, которое может быть перехвачено, содержит информацию о диалоговых окнах, возвращаемую из структуры CPLINFO. Сообщение CPL_NEWINQUIRE не должно перехватываться, поскольку оно может снизить производительность. Если вы видите, что загрузка панели управления происходит медленно, то причина этого может лежать в отправке другими аплетами сообщения CPL_NEWINQUIRE. Сообщение CPL_NEWINQUIRE совпадает с сообщением CPL_INQUIRE, за исключением того, что lParam2 является указателем на структуру NEWCPLINFO. Это сообщение полезно только в том случае, если вам нужно изменить пиктограмму вашего аплета или отобразить информацию, зависящую от состояния вашей системы.

Однако аплет панели управления должен обрабатывать оба сообщения: CPL_INQUIRE и CPL_NEWINQUIRE, с учетом того, что сообщение CPL_NEWINQUIRE можно проигнорировать. Разработчик не должен делать никаких допущений насчет порядка или зависимости CPL_INQUIRE-сообщений. lParam1 — регистрируемый номер аплета — значение, лежащее в диапазоне от 0 до (CPL_GETCOUNT - 1). lParam2 — указатель на структуру CPLINFO.

Функция CPLApplet() после этого получает сообщение CPL_DBCLK, означающее, что пользователь активизировал аплет панели управления. Это сообщение содержит идентификатор диалогового окна и значение, посылаемое для параметра lData. Когда управляющее приложение посылает сообщения CPL_DBLCLK и CPL_STOP, это значение передается назад вашему приложению.

После отправки последнего сообщения CPL_STOP функция CPLApplet() получает сообщение CPL_EXIT. Эта функция должна освободить всю память, выделенную для аплета. Немедленно по окончании этого процесса вызывается функция FreeLibrary().

Создание аплета панели управления

Пришло время самим создать аплет. Но прежде следует ознакомиться с общими правилами, которые нужно соблюдать при создании аплета панели управления.

- Готовый аплет (которым можно пользоваться) нужно скопировать в папку \windows\system или \winnt\system32. Все аплеты панели управления хранятся именно здесь.

- Аплет панели управления можно зарегистрировать в системном реестре. Для этого можно использовать ключ MMCPL под ключевым полем HKEY_CURRENT_USER\Control Panel key.
- Не забудьте переименовать скомпилированную DLL-библиотеку в файл с расширением .CPL до его копирования в одну из указанных папок. В C++Builder можно установить такое расширение по умолчанию с помощью команды Project⇨Options (Проект⇨Параметры). Выберите в диалоговом окне Project Options (Параметры проекта) вкладку Application (Приложение). В разделе Output Settings (Внешние параметры) поле Target Extension (Расширение результата) можно установить равным CPL. После этого по окончании компиляции скомпилированной программе будет присвоено расширение .CPL.
- Утилита CONTROL.EXE не гарантирует соблюдение порядка загрузки DLL-аплетов. Аплеты можно отсортировать в целях отображения. Иногда их загрузку можно выполнить в алфавитном порядке, но это определяется самим управляющим приложением или панелью управления.

Приняв эти правила к сведению, можно переходить к созданию аплета.

1. Откройте C++Builder.
2. Закройте все окна и не сохраняйте никакие проекты, формы и т.д. Убедитесь, что в C++Builder не загружены никакие проекты.
3. Выберите команду File⇨Close All (Файл⇨Закрыть все). Наша задача — создать две формы с несколькими компонентами. Это будет основа аплета панели управления.
4. Выберите команду File⇨New Form (Файл⇨Создать форму) из меню File окна C++Builder.
5. На экране появится форма. Поместите в нее надпись и кнопку. Установите свойства, показанные в табл. 24.30.

Таблица 24.30. Свойства компонентов формы для примера аплета

Компонент	Свойство
Label1	Caption = This is Dialog 1 in my applet!
Button1	Caption = Close

6. Введите в обработчик события OnClick кнопки следующий код.

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

7. Сохраните форму под именем Form1.
8. Выберите из меню File команду File⇨New Form и разместите в новой форме те же самые компоненты.
9. Добавьте для кнопки Button1 событие OnClick и поместите в его обработчик код, приведенный в пункте 6.
10. В окне инспектора объектов присвойте форме имя Form2 и сохраните форму как Form2.
11. На данном этапе вы должны иметь две отдельные формы. Выберите из меню File C++Builder команду File⇨Close All.

Теперь необходимо создать файл ресурсов для пиктограмм, используемых аплетом. Необязательно создавать две пиктограммы, но при отсутствии вашей пиктограммы Windows по умолчанию установит "свою"; более того, обе части аплета в этом случае получат одну и ту же пиктограмму. А поскольку вам, скорее всего, захочется, чтобы аплет чем-то выделялся, то лучший способ — создать свой вариант пиктограммы, отличный от стандартного.

Для этого сверните окно C++Builder и откройте редактор изображений Image Editor. Создайте новый файл ресурсов. Это можно сделать, выбрав команду File⇨New (Файл⇨Создать) с последующим выбором варианта Resource File (Файл ресурсов). Появится диалоговое окно. Щелкните внутри этого диалогового окна правой кнопкой мыши и из контекстного меню выберите команду New⇨Icon (Создать⇨Пиктограмму). Выберите вариант 32×32, 16 цветов и щелкните на кнопке OK. Щелкните правой кнопкой мыши на новой пиктограмме и переименуйте ее, используя имя `ICON1`.

Затем выберите родительское дерево `CONTENTS`, щелкните на нем правой кнопкой мыши и выберите команду New⇨Icon. Выберите вариант 32×32, 16 цветов и щелкните на кнопке OK. Щелкните правой кнопкой мыши на новой пиктограмме и переименуйте ее, используя имя `ICON2`. Теперь у вас должно быть две пиктограммы с именами `ICON1` и `ICON2`.

Сохраните файл ресурсов под именем `MyCPL.RES`.

Теперь создадим для пиктограмм `ICON1` и `ICON2` рисунки. Для этого дважды щелкните на их заголовках. Нарисуйте что хотите. В этом примере для пиктограмм был использован текст `i1` и `i2` соответственно. Вы можете поступить по-своему.

По завершении работы выберите команду File⇨Save (Файл⇨Сохранить). Закройте редактор изображений. Запомните, в какой папке вы сохранили файл ресурсов.

Итак, мы создали два диалоговых окна, которые появятся в нашем аплете, и пиктограммы, сопровождающие эти диалоговые окна, но мы не создали пока самого аплета. Для этого сверните окно C++Builder и закройте все, что осталось открытым. Затем выберите команду File⇨New (Файл⇨Создать). Откроется диалоговое окно New Items (Новые элементы). Вы "попали" в хранилище объектов (Object Repository), и оно будет содержать элементы, которые вы выберете. Выберите во вкладке New Page (Новая страница) пиктограмму DLL Wizard (Мастер DLL). Откроется диалоговое окно DLL Wizard, предлагая вам указать тип создаваемой библиотеки DLL. Вполне подойдут значения, предложенные по умолчанию, поэтому щелкните на кнопке OK, чтобы выбрать стандартные параметры. Через несколько секунд C++Builder сгенерирует код, необходимый для создания библиотеки DLL.

```
//---------------------------------------------------------------------

#include <vcl.h>
#include <windows.h>
#pragma hdrstop
#pragma argsused
int WINAPI DllEntryPoint(HINSTANCE hinst, unsigned long reason,
                                                  void* lpReserved)
{
        return 1;
}
```

Теперь мы готовы к созданию аплета.

Прежде чем продолжить, не забудьте сохранить свою работу. Выберите команду File⇨Save Project As (Файл⇨Сохранить проект как) и сохраните модуль `Unit1.cpp` как `Main`. Сам проект сохраните под именем `MyCPL`, желательно в той же папке, где хранятся недавно созданные вами формы. А теперь вернемся в C++Builder.

Воспользуйтесь командой Project⇨Add to Project (Проект⇨Добавить к проекту) и выберите из списка файлов Form1. Сделайте то же самое, чтобы добавить форму Form2.

Если вы все сделали правильно, то у вас должны сейчас отображаться обе формы, и обе они должны быть добавлены в проект. В этом можно убедиться, выбрав команду View⇨Project Manager (Вид⇨Менеджер проекта). Список должен включать как Form1, так и Form2.

Теперь нужно добавить в проект файл ресурсов, содержащий пиктограммы. Выберите для этого команду Project⇨Add to Project (Проект⇨Добавить к проекту). Появится список файлов. В раскрывающемся списке File Type (Тип файла) выберите для файлов ресурсов, вместо типа .cpp, тип .RES. В списке файлов должен появиться файл MyCPL.RES. Выберите этот файл. Это значит, что он будет добавлен к нашему проекту.

Прежде чем продолжить работу, убедитесь, что скомпилированный файл библиотеки DLL имеет расширение .CPL. Выберите команду Projects⇨Options⇨Application Page (Проект⇨Параметры⇨Приложение). В поле Output File Settings замените расширение .DLL расширением .CPL. Когда C++Builder скомпилирует нашу DLL-библиотеку, результирующий файл будет иметь тип аплета панели управления, т.е. расширение .CPL.

Следующий этап состоит в реализации функции CPLApplet() в исходном .cpp-файле, как показано ниже. Не забудьте включить заголовочный файл CPL.H.

```
CPlApplet(HWND HwControlPanel, int Msg, int LParam1, int LParam2)
{
  switch(Msg)
   {
    case CPL_INIT:
      // Возвращает признак успешной инициализации...
      // Жив ли он еще, да и вообще, аплет ли это?
       MyProgInstance = GetModuleHandle("MyCPL.cpl");
       return true;

    case CPL_GETCOUNT:
      // Сколько пиктограмм должно быть отображено в аплете?
      return 2;

    case CPL_NEWINQUIRE:
      {
        // Панель управления запрашивает информацию для
        // отображения...Эта информация считывается во время
        // загрузки панели управления...

         switch (LParam1)
         {
          case 0:
            {
               NEWCPLINFO *Info = (NEWCPLINFO *)LParam2;
               ZeroMemory(Info, sizeof(NEWCPLINFO));
               Info->dwSize = sizeof(NEWCPLINFO);
               Info->hIcon = LoadIcon(HInstance, "ICON1");
               strcpy(Info->szName, "The Applet #1");
               strcpy(Info->szInfo, "This is Applet #1");
                return 0;
```

```
          }
        case 1:
          {
             NEWCPLINFO *Info2 = (NEWCPLINFO *)LParam2;
             ZeroMemory(Info2, sizeof(NEWCPLINFO));
             Info2->dwSize = sizeof(NEWCPLINFO);
             Info2->hIcon = LoadIcon(HInstance, "ICON2");
             strcpy(Info2->szName, "The Applet #2");
             strcpy(Info2->szInfo, "This is Applet #2");
              return 0;
          }
       }

        // Если нужны еще аплеты...

      }

   case CPL_DBLCLK:
    // Поскольку был двойной щелчок на отображенной пиктограмме,
    // нужно показать форму!
       try
       {      //!

         if(LParam1==0)
          {
           Application->Initialize();
           Application->CreateForm(__classid(TForm1), &Form1);
           Application->Run();
          }
          else
          {
           Application->Initialize();
           Application->CreateForm(__classid(TForm2), &Form2);
           Application->Run();
          }
       }
       catch (Exception &exception)
       {
         Application->ShowException(&exception);
       }
       return 0;

    case CPL_STOP: break;
    case CPL_EXIT: break;

  } //Конец блока switch
}
```

Текст этого примера программы должен быть аналогичен коду, приведенному в листинге 24.13.

Листинг 24.13. Создание основной части программы аплета панели управления

```
//---------------------------------------------------------------
#include <vcl.h>
#include <cpl.h>       // Не забудьте включить этот заголовочный
                       // файл для CPL-библиотек!

#include <windows.h>
#pragma hdrstop
#pragma argsused

// Велите C++Builder использовать все эти ресурсы...
USEFORM("Form1.cpp", Form1);
USEFORM("Form2.cpp", Form2);
USERES("MyCpl.res");

// Объявляем дескриптор нашего аплета.
HINSTANCE MyProgInstance =NULL;

// Функция DLLEntryPoint. Начало DLL....
int WINAPI DllEntryPoint(HINSTANCE hinst, unsigned long reason, void*
BlpReserved)
{
  if (reason==DLL_PROCESS_ATTACH)
      MyProgInstance=hinst;
         return 1;
}
//---------------------------------------------------------------


// Панель управления будет делать все вызовы и запросы с помощью
// этой функции. Как видите, функция WINMAIN заменена другой....
extern "C" int __stdcall __declspec(dllexport)

CPlApplet(HWND HwControlPanel, int Msg, int LParam1, int LParam2)
{
  switch(Msg)
   {
    case CPL_INIT:
      // Возвращает признак успешной инициализации...
      // Жив ли он еще, да и вообще, аплет ли это?

       MyProgInstance = GetModuleHandle("MyCPL.cpl");
       return true;

    case CPL_GETCOUNT:
      // Сколько пиктограмм должно быть отображено в аплете?
      return 2;

    case CPL_NEWINQUIRE:
      {
```

```
    // Панель управления запрашивает информацию для
    // отображения...Эта информация считывается во время
    // загрузки панели управления...

     switch (LParam1)
     {
      case 0:
        {
           NEWCPLINFO *Info = (NEWCPLINFO *)LParam2;
           ZeroMemory(Info, sizeof(NEWCPLINFO));
           Info->dwSize = sizeof(NEWCPLINFO);
           Info->hIcon = LoadIcon(HInstance, "ICON1");
           strcpy(Info->szName, "The Applet #1");
           strcpy(Info->szInfo, "This is Applet #1");
            return 0;
         }
      case 1:
        {
           NEWCPLINFO *Info2 = (NEWCPLINFO *)LParam2;
           ZeroMemory(Info2, sizeof(NEWCPLINFO));
           Info2->dwSize = sizeof(NEWCPLINFO);
           Info2->hIcon = LoadIcon(HInstance, "ICON2");
           strcpy(Info2->szName, "The Applet #2");
           strcpy(Info2->szInfo, "This is Applet #2");
            return 0;
        }
     }

      // И другие аплеты, если нужно...

   }

case CPL_DBLCLK:
 // Поскольку был двойной щелчок на отображенной пиктограмме,
 // нужно показать форму!
    try
    {      //!

      if(LParam1==0)
       {
        Application->Initialize();
        Application->CreateForm(__classid(TForm1), &Form1);
        Application->Run();
       }
       else
       {
        Application->Initialize();
        Application->CreateForm(__classid(TForm2), &Form2);
        Application->Run();
       }
```

```
        }
        catch (Exception &exception)
        {
          Application->ShowException(&exception);
        }
        return 0;

     case CPL_STOP: break;
     case CPL_EXIT: break;

  } // Конец блока switch
}
```

Верите или нет, но с программой покончено! Вам осталось сохранить свою работу, скомпилировать и скомпоновать проект. По завершении поместите файл `MyCPL.CPL` в папку `\windows\system` или `\windows\system32`. Открыв панель управления, вы должны увидеть только что "испеченный" аплет.

Чтобы быстро запустить наш аплет без вмешательства панели управления, можно выполнить следующую команду, используя меню Start⇨Run (Пуск⇨Выполнить).

```
rundll32 shell32.dll,Control_RunDLL mycpl.cpl @0
```

Здесь `@` описывает, какое диалоговое окно аплета вы собираетесь открыть (их счет начинается с нуля). Например, приведенная выше строка кода активизирует первое диалоговое окно аплета.

А следующая команда откроет второе диалоговое окно аплета.

```
rundll32 shell32.dll,Control_RunDLL mycpl.cpl @1
```

Утилита `RunDLL32.EXE` вызывает библиотеку `Shell32.DLL`, которая содержит специальную функцию для вызова различных аплетов с заданием параметров.

Возможно, вы заметили в примере программы использование сообщения `CPL_NEWINQUIRE`, и вам интересно, почему вместо сообщения `CPL_INQUIRE` было включено сообщение `CPL_NEWINQUIRE`. Это было сделано ради краткости примера и простоты описания. Структуры сообщений `CPL_NEWINQUIRE` и `CPL_INQUIRE` различны. Если используется сообщение `CPL_INQUIRE`, пиктограммы и строки для имен аплетов должны содержаться внутри файла сценария ресурсов (с расширением `.rc`). `.rc`-файл можно скомпилировать с компилятором ресурсов C++Builder `BRCC32.EXE`, который превращает его в файл ресурсов (с расширением `.RES`). Убедитесь в существовании заголовочного файла для определения значений текстовых констант.

Ниже приводится пример создания списка строк.

```
STRINGTABLE
 BEGIN
ID_ICONTEXT1 "Это аплет 1";
ID_ICONTEXT2 "Аплет №2";
 END
```

Для использования сообщения `CPL_INQUIRE` требуется, чтобы в приложении применялась функция `LoadString()` для получения строк списка, который затем назначался этой структуре. Это позволяет аплету использовать список строк и ресурсы из пиктограмм. Однако на это требуется немного больше работы, чем при использовании структуры `CPL_NEWINQUIRE`. Поскольку здесь представлен простой пример программы, то от использования структуры

CPL_NEWINQUIRE потери в производительности несущественны. Однако при создании большой программы необходимо предусмотреть использование структуры CPL_INQUIRE. Как было продемонстрировано, создание небольших простых и ясных приложений, которые работают под "присмотром" панели управления, — вполне достижимая и несложная задача.

Резюме

В этой главе мы рассмотрели много элементов и аспектов интерфейса Win32 API. Используя в своих программах функции Win32 API, разработчики начинают понимать сложные элементы нижнего уровня Windows. А понимание интерфейса Win32 API обеспечивает значительные преимущества для разработчика приложений Windows.

Применение интерфейса Win32 API в приложении может быть сопряжено с определенными трудностями, но их преодоление всегда интересно. Эта работа напоминает задачу соединения фрагментов головоломки, чтобы получить нужный результат: если вы правильно соединяете части, у вас получится превосходная картинка. Один из ключевых моментов эффективной разработки в среде C++Builder — понимание, когда именно стоит использовать интерфейс Win32 API. Например, когда для выполнения задачи нет доступных элементов библиотеки VCL, следует рассмотреть вопрос об использовании Win32 API. Это также следует сделать, если функции Win32 API повысят производительность и расширят возможности приложения.

В этой главе приведено несколько примеров, которые продемонстрировали, как и когда следует использовать обычные функции, функции обратного вызова и структуры Win32 API. Надеемся, что это пополнило арсенал ваших знаний, но нужно понимать, что рассмотренные здесь темы — лишь скольжение по поверхности обширного водоема, в глубине которого скрыто огромное богатство и разнообразие. Имейте в виду, что в среде Borland C++Builder предусмотрена возможность создания новых библиотек VCL, а также могучих приложений на основе как популярных, так и редко используемых функций Win32 API. Разработчик должен всегда держать в уме возможность использования Win32 API (если не в данном, то в следующем своем приложении Windows). Рано или поздно, но вы обязательно обнаружите, что с помощью интерфейса Win32 API можно воплотить в жизнь ваши лучшие идеи.

Попутного вам ветра!

Глава

25

Методы использования мультимедиа

Дэймон Чандлер
Сью-Фэн Ву
Роб Аллен

Многим приложениям требуется мультимедийная поддержка определенного типа. Довольно часто мультимедийные технологии используются просто для улучшения пользовательского интерфейса приложения. В некоторых случаях приложения опираются на мультимедийно-ориентированные методы обмена информацией с пользователем, т.е. иногда приложения специально разрабатываются для создания, управления, отображения или воспроизведения мультимедийных файлов.

В этой главе мы рассмотрим несколько методов поддержки мультимедиа в приложениях. Степень поддержки основана исключительно на конкретных потребностях самого приложения. Windows предоставляет (об этом речь впереди) несколько интерфейсов; они обеспечивают общие средства, посредством которых приложение может общаться с драйвером принимающего устройства.

Поддержка отображения неподвижных картинок осуществляется с помощью графического интерфейса Windows с устройствами (Graphics Device Interface — GDI). Приложение косвенно связывается с драйвером дисплея посредством функций, обеспечиваемых GDI. Здесь мы рассмотрим как сам интерфейс GDI, так и соответствующие ему VCL-классы. Если говорить более конкретно, мы рассмотрим использование таких классов, как `TCanvas`, `TBrush`, `TPen` и `TFont`, а также характер использования объектов этих классов для воспроизведения графических изображений на мониторе и принтере. Затем мы уделим внимание VCL-классам `TBitmap` и `TJPEGImage` и их использованию для поддержки растровых и JPEG-изображений, соответственно. Мы также рассмотрим основы других форматов файлов изображений и некоторые методы поддержки этих типов изображений в VCL-приложении.

Затем мы перейдем к обработке изображений, продемонстрировав несколько методов манипуляции ими. В состав описанных здесь методов входят: доступ к значениям пикселей и их модификация, "пороговая" обработка, инвертирование (обращение) цветных и ахроматических изображений, коррекция контраста с помощью гистограмм, масштабирование с использованием геометрических преобразований, а также выравнивание и выделение краев (контуров) с помощью пространственных операций.

Аудио- и видеоподдержка обычно осуществляется посредством *интерфейса для аппаратно-независимого управления событиями в системе мультимедиа* (Media Control Interface — MCI). В этом случае приложение косвенно связывается с драйверами устройств воспроизведения аудио- и видеоданных с помощью функций, предусмотренных в MCI. Кроме того, Windows поддерживает более специализированные интерфейсы для специфических медиа-форматов. Мы продемонстрируем использование MCI-интерфейса для воспроизведения аудио- и видеоданных, а также использование интерфейса Waveform-Audio Interface для воспроизведения звуковых (waveform-audio) файлов, полученных в результате восстановления волновой формы сигнала из дискретизированной.

Графический интерфейс с устройствами (GDI)

Большинство "надводной", т.е. видимой части GUI-программ Windows, написанных в среде C++Builder, строится с использованием форм и элементов управления. Однако некоторые из них включают графические изображения и тем самым вторгаются на "территорию", которая принадлежит GDI-программам Windows. GDI — одна из центральных частей операционной системы Windows, которая "расквартирована" в библиотеках `GDI.DLL` и `GDI32.DLL`. Она охватывает сотни функций (например, в моей системе Windows NT библиотека `GDI32.DLL` насчитывает 401 функцию). И вообще, все рисующие компоненты Windows используют GDI, включая саму Windows.

Основная отличительная черта интерфейса GDI — независимость от конкретных устройств. При рисовании с помощью функций GDI вам совсем не обязательно знать специфику программирования каждой видеокарты или принтера, предлагаемых рынком. GDI обеспечивает некоторый уровень абстракции между кодом вашего приложения и аппаратными средствами.

Тесные рамки одной главы не позволяют раскрыть все многообразие средств, доступных в GDI, поэтому обращайтесь к справке по VCL, начиная с класса `TCanvas`. Кроме того, существует большой объем информации в справочной системе Microsoft, к которой можно получить доступ, выбрав команду Start⇨Help⇨Windows SDK (Пуск⇨Справка⇨Оборудование и программное обеспечение).

Интерфейс Windows API и контекст устройства

Для взаимодействия с GDI, как и с любой частью программирования Windows, вам понадобится дескриптор. В данном случае он называется *контекстом устройства* (Device Context — DC). В стандартном интерфейсе Windows API получить его можно с помощью следующего оператора.

```
HDC hDC = GetDC(hWindow);
```

Возвращаемый контекст устройства привязывается к конкретному окну, и вне клиентской области этого окна рисовать нельзя. Чтобы нарисовать где-нибудь на рабочем столе, в качестве аргумента `hWindow` функции `GetDC()` нужно передать значение `NULL`. Неудивительно, что в программировании Windows API контекст устройства (DC) передается как параметр в каждую GDI-функцию рисования. Завершив рисование, следует освободить DC с помощью следующей функции.

```
ReleaseDC(hD);
```

Понятие о классе `TCanvas`: интерфейс с C++Builder

В программировании в среде C++Builder уже проверено, что для повышения эффективности предпочтительнее использовать компоненты, а не исходный API-код. Причем неважно, что рисовать: линии или фигуры с заливкой каким-нибудь цветом, поэтому мы используем компонент `TCanvas`. `TCanvas` представляет собой оболочку, “нашпигованную” GDI-функциями, которая доступна в качестве свойства компонентов `TForm` и `TPrinter` и большинства популярных элементов управления. Например, чтобы нарисовать прямоугольник в форме, используйте следующую строку.

```
Canvas->Rectangle(10, 10, 100, 100);
```

В чем суть компонента `TCanvas`

Компонент `TCanvas` обеспечивает объектно-ориентированный подход к программированию Windows API. Самая большая его заслуга в том, что он обрабатывает ресурсы за вас. Существуют довольно сложные правила относительно того, как управлять ресурсами GDI при использовании оригинальных API-функций; а компонент `TCanvas` незаметно для вас выполняет эти правила. Кроме того, `TCanvas` обеспечивает стандартную систему свойств VCL для получения и установки множества атрибутов соответствующего контекста устройства, что, естественно, заведомо упрощает создаваемый код и делает его намного понятнее и читабельнее.

Основные методы

Компонент TCanvas имеет несколько свойств, о которых вам необходимо знать:

- TPen Pen – выбранное в данный момент перо;
- TBrush Brush – выбранная в данный момент кисть;
- TFont Font – выбранный в данный момент шрифт;
- TPoint PenPos — позиции, в которых перо может рисовать.

Важно иметь в виду, что канва при рисовании будет использовать текущее перо, кисть, шрифт и позицию. Поэтому перед вызовом функции рисования их значения нужно соответствующим образом изменить (установить).

Что упущено в реализации компонента TCanvas

Компонент TCanvas и связанные с ним классы запрограммированы в файле graphics.pas, но они не охватывают всех GDI-функций. В среде Borland реализовано только большинство обычно используемых функций. Кроме того, некоторые вспомогательные классы (например, TRect и TPoint) не обеспечивают более широкие возможности, чем лежащие в их основе Windows API-аналоги, и могли быть более полезными для программиста.

Но еще не все потеряно. В компоненте TCanvas предусмотрено свойство Handle, чтобы разрешить доступ к функциям GDI. Возвращаясь к нашему прямоугольнику, его можно построить и так.

```
Rectangle(Canvas->Handle, 10, 10, 100, 100);
```

В обоих случаях будут получены одинаковые результаты.

На заметку

Некоторые классы VCL, используемые для включения популярных GDI-структур, не становятся намного более полезными, чем их "начинка". Красноречивыми примерами таких классов являются TRect, TPoint и TSize. Их можно назвать слишком тонкими классами-оболочками. Инженеры компании Borland могли бы больше вложить в эти вспомогательные классы.

Например, компонент TRect представляет собой инкапсуляцию прямоугольника. Он содержит четыре полезных функции: operator==(), operator!=(), Height() и Width(). Будь у разработчика чуть больше воображения, этот компонент можно было бы сделать по-настоящему полезным. Например, с помощью функций этого компонента можно было бы предоставить возможность выяснить, находится ли некоторая точка или прямоугольник внутри данного прямоугольника или касается его. Кроме того, очень желательно иметь возможность масштабирования прямоугольника или его перемещения.

Трудно ли обеспечить собственный вариант компонента? Конечно нет, но всегда удобнее пользоваться уже готовыми вещами, не так ли? Например, если бы уже существовала функция определения включения точки в область прямоугольника, то достаточно было бы записать следующее.

```
If( myRect->Contains(myPoint){  // какие-нибудь действия  }
```

Использование компонента TCanvas

Обычно рисуют только на канве в рамках обработчика события OnPaint формы TForm. Это гарантирует, что сохраняется текущее отображение. Также нередко создается скрытый вариант канвы с помощью компонента TImage, который имеет тот же размер, что и форма, и

рисование происходит на скрытой канве, которая затем копируется в канву формы в событии OnPaint. Для такого варианта рисования характерно преимущество ускорения отображения.

В качестве примера рассмотрим реализацию аналоговых часов. При отсутствии секундной стрелки обновлять их изображение нужно только раз в минуту. В процедуре рисования необходимо вычислять угол отклонения стрелок, что займет некоторое время. Когда окно будет загораживаться чем-то другим и снова открываться для обозрения, будет вызываться функция OnPaint. Для оптимизации скорости перерисовки используется вторая скрытая канва.

Обработчик события OnPaint должен иметь примерно такой вид.

```
void __fastcall TForm1::FormPaint(TObject *Sender)
{
    // Копируем информацию со скрытой канвы в канву формы.
    Canvas->CopyRect(ClientRect, HiddenImage->Canvas, HiddenImage->ClientRect);
}
```

К этому методу мы еще вернемся в другом примере ниже в этом разделе.

Рисование линий и окружностей

С помощью компонента TCanvas рисование линий и окружностей — проще пареной репы. Для этого достаточно установить нужное перо, а затем вызвать соответствующую функцию. В компоненте TCanvas предусмотрено множество функций рисования линий, описанных в документации C++Builder. Для того чтобы нарисовать простую линию, достаточно использовать две команды.

```
Canvas->PenPos = TPoint(1, 1);
Canvas->LineTo(9, 1);
```

Обратите внимание, что все функции *XxxTo()* начинают работать с текущей позиции пера и, как и большинство GDI-функций рисования, *не включают последнюю заданную точку*. Это означает, что при рисовании прямоугольника из позиции (1,1) до позиции (10,10) пиксель в позиции (10,10) не рисуется. На рис. 25.1 показано, как линия, нарисованная из точки (1,1) до точки (9,1), не заполняет последний пиксель. Так же работают и такие функции, как Ellipse(), для которой предварительно задается ограничительный прямоугольник.

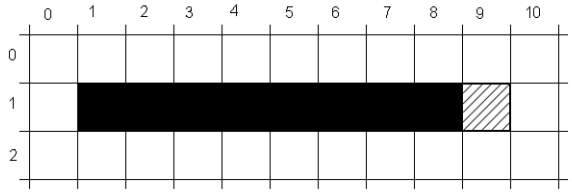


Рис. 25.1. Функция LineTo(9,1) не заполняет последний пиксель

Создание заполненных областей

Для создания заполненных (цветом) изображений используется как перо, так и кисть. Перо применяется для рисования контура, а кисть — для заполнения интерьера. Например, нарисуем прямоугольник следующим образом.

```
TRect MyRect(10, 10, 100, 300);
Canvas->Rectange(myRect)
```

Рисование эллипсов немного отличается тем, что к ним нужно задавать ограничительный прямоугольник. Затем система рисует эллипс так, чтобы он касался сторон этого прямоугольника.

Вывод текста

Существует несколько способов вывода текста. Самый простой из них — использовать функцию `TextOut()`, которая начинает работу с заданной позиции `(x,y)`. Эта функция изменяет значение `PenPos` в соответствии с концом выведенного текста, облегчая потенциальное продолжение этой операции. Функция `TextRect()` выведет текст внутри заданного прямоугольника и отсечет любой не поместившийся в него текст.

Размер текста зависит от шрифта. При использовании равноширинного шрифта (с фиксированным шагом), например Courier, каждая буква занимает одинаковое пространство, будь то *i* или *w*. И, наоборот, для шрифта с переменным шагом все буквы, имея различную ширину, занимают различное пространство: поэтому *i* занимает меньше места, чем *w*. Чтобы вычислить, какую ширину и высоту займет строка, в компоненте `TCanvas` предусмотрена функция `TextExtent()`, возвращающая значение типа `TSize`, которое можно использовать для адекватного размещения текста.

В примере с аналоговыми часами текст выводится с помощью следующего кода.

```
AnsiString text = "Часы для меню...";
TSize textSize = HiddenImage->Canvas->TextExtent(text);
int x = (HiddenImage->Width - textSize.cx) / 2;
int y = HiddenImage->Height - textSize.cy - 2;
HiddenImage->Canvas->TextOut(x, y, text);
```

При выполнении этого кода выведенный текст будет отцентрирован в нижней части объекта `HiddenImage`.

Настройка рисунка

Линии и текст не будут иметь приличного вида, если вы не сможете настроить свой рисунок. Чаще всего приходится менять цвет и толщину линий, а также шрифт текста. C++Builder обеспечивает ряд вспомогательных классов для компонента `TCanvas`, которые позволяют легко выполнить такую настройку.

На заметку Чтобы настроить конкретную операцию рисования, важно, чтобы необходимое изменение (например, цвета) делалось *до* вызова функции рисования.

TColor

Цвет графического объекта в библиотеке VCL устанавливается с помощью свойства `TColor`. `TColor` представляет собой VCL-отображение Windows API-значения `COLORREF`, которое для задания цвета использует 32-разрядное число. Любой цвет состоит из красного, зеленого и синего компонентов, поэтому красный соответствует значению `(255,0,0)`, а белый — `(255,255,255)`. Этот принцип трехцветности заложен в механизм отображения цветов стандартным цветным монитором; если очень внимательно посмотреть на экран через увеличительное стекло, вы увидите три отдельных пикселя (красный, зеленый и синий) для каждого пикселя, который может видеть ваш глаз с некоторого расстояния.

Простые арифметические расчеты показывают, что при такой организации максимально возможное количество цветов составляет 255×255×255 = 16 581 375, или приблизительно

16 миллионов. Не все дисплеи, подключенные к Windows-компьютеру, способны отображать так много цветов при любом заданном разрешении, поэтому возможное количество отображаемых цветов может выражаться такими числами, как 16, 256, 65536 или 16 миллионов. Количество доступных цветов называют *глубиной цвета*. В табл. 25.1 перечислены возможные цвета и дано соответствующее описание.

Таблица 25.1. Представление цветов при различных глубинах цвета

Количество цветов	Описание
16 цветов	При использовании 16-цветного дисплея VGA цвета фиксированы и перечислены в справке для свойства `TColor`
256 цветов	Работа 256-цветного дисплея опирается на использование палитр. Это означает, что одновременно может отображаться только 256 цветов, но при этом из всего диапазона цветов можно выбрать, какие именно цвета будут отображаться
65 536 цветов	Красно- (Red), зелено- (Green), синие (Blue) значения (RGB-значения) хранятся в виде 16-разрядных чисел. При выборе цвета Windows применяет значение, которое ближе всего подходит к выбранному
16 миллионов цветов	Для отображения доступны все цвета

Рамки этого раздела не позволяют раскрыть тему управления палитрами для 256-цветных дисплеев. Достаточную информацию можно получить из такого издания, как *Programming Windows*, Microsoft Press, ISBN: 157231995X. Чтобы определить количество доступных цветов, используйте Windows API-функцию `GetDeviceCaps()`, которая может сообщить количество матриц (plane) и число бит на пиксель. Следующий фрагмент кода позволит вычислить количество цветов.

```
int BitsPerPixel = GetDeviceCaps(Canvas->Handle, BITSPIXEL);
int NumberOfPlanes = GetDeviceCaps(Canvas->Handle, PLANES);

int NumberofColours = 1 << (NumberOfPlanes * BitsPerPixel);
```

`TPen`

При рисовании линий или таких контурных объектов, как окружность, для определения цвета, толщины и стиля используется текущее перо. Стиль определяет, какая именно рисуется линия: сплошная, штриховая, пунктирная и т.д. Если ширина больше 1, линия будет сплошной, поэтому, если вам нужно нарисовать толстую штриховую линию, нарисуйте несколько таких линий подряд.

Компонент `TPen` имеет режим, который определяет, как на цвет пера влияет цвет, уже существующий на канве. Особого внимания заслуживает режим `pmNotXor`, который выполняет операцию “не исключающее ИЛИ” с цветом пера и цветом канвы. В частности, это удобно при рисовании “временных” линий поверх изображения с намерением вскоре эти линии стереть. Стирание такой линии можно осуществить путем ее перерисовки поверх первой линии в режиме `pmNotXor`, который позволит нейтрализовать обе линии, оставив для отображения исходное изображение. Этот метод можно использовать, например, для выбора области масштабирования в программе, работающей с графикой. По мере перемещения мыши вы видите, как изменяется ограничительный прямоугольник, создающий область масштабирования, а исходное изображений при этом сохраняет тот же вид.

TBrush

Для определения, как (чем) заполнить (залить) объект, используется текущее состояние кисти. Компонент TBrush имеет свойства Color и Style. Свойство Style определяет, заливать ли объект или использовать некоторый узор для заполнения области. Свойство Style может принимать следующие значения: bsSolid, bsCross, bsClear, bsDiagCross, bsBDiagonal, bsHorizontal, bsFDiagonal и bsVertical. Результат применения этих стилей можно увидеть в примере программы создания аналоговых часов.

TFont

Текущий шрифт используется в таких текстовых функциях, как TextOut(). Шрифт имеет все необходимые атрибуты: цвет, имя, высоту и стиль. Размер шрифта можно установить двумя путями: с помощью свойства Height в пикселях или свойства Size в пунктах. Достаточно установить одно свойство — второе будет вычислено автоматически. Такой "плюрализм" весьма полезен, поскольку большинство пользователей предпочитает устанавливать размер шрифта в пунктах, а не в пикселях.

Пример создания аналоговых часов

Для демонстрации этих идей подготовлен проект аналоговых часов, clock.bpr, который можно найти в папке GDI на компакт-диске, прилагаемом к книге. Этот вариант часов основан старом Borland C++ OWL-примере aclock и отображает простые часы с часовой и минутной стрелками (рис. 25.2).

Все рисование выполняется на канве скрытого компонента TImage, содержимое которого копируется на Canvas формы в процессе выполнения обработчика события OnPaint(). В этом примере при рисовании линий для стрелок часов и эллипса для циферблата демонстрируется использование таких свойств канвы, как Font, Brush и Pen. Используя контекстное меню, можно изменить стиль рисования стрелок и стиль кисти для циферблата, чтобы показать возможность применения различных эффектов.

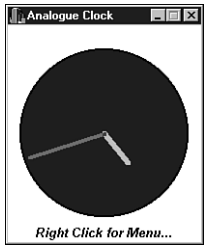


Рис. 25.2. Пример программы создания аналоговых часов

На заметку

Цель этой программы — показать использование объекта Canvas для рисования. Важными функциями этого свойства формы являются InitialiseImage(), DrawClockToHiddenImage() и FormPaint(). Поскольку это всего лишь пример, в нем (ради краткости) отсутствует контроль ошибок, а сам текст программы не претендует на звание лучшей программы тысячелетия.

Поддержка изображений

Интерфейс Windows GDI обеспечивает встроенную поддержку растрового формата файла и предлагает множество GDI-функций, которые специально написаны для обработки растровых объектов. Однако если вы заинтересованы в отображении файлов, использующих другие форматы изображений, вам придется применять другие средства. В частности, вам понадобится процедура преобразования, с помощью которой вы сможете построить растровый объект на основе информации, содержащейся в самом файле.

Мы здесь не ставим цель углубиться в особенности каждого формата файла изображения. Для этого потребовалась бы отдельная книга. Мы же ограничимся общим обзором самых популярных форматов файлов. Сначала остановимся на растровом объекте, соответствующих ему GDI-функциях и структурах, а также на VCL-классе `TBitmap`. Затем рассмотрим другие форматы файлов изображений. Не оставим без внимания и библиотеки сторонних производителей, которые используются для преобразования растровых изображений.

Растровый объект Windows

Интерфейс GDI работает с двумя типами растровых изображений: аппаратно-зависимыми растрами (device-dependent bitmaps — DDB) и аппаратно-независимыми растрами (device-independent bitmaps — DIB). Первый из них представляет собой тип графического объекта, который определяется структурой `BITMAP`, и его можно использовать практически так же, как и большинство других графических объектов. Например, можно выбрать DDB-объект в контексте устройства памяти и применить к нему любые подходящие GDI-функции отображения. К сожалению, интерфейс GDI не обеспечивает прямых средств доступа к битам DDB-объекта. DIB-объект, наоборот, определяется информацией, содержащейся в структуре `BITMAPINFO`, и представляет собой массив пикселей. В этом случае вы всегда имеете прямой доступ к битам DIB-объекта. Однако здесь можно попасть в ловушку: DIB-объекты нельзя выбрать в контексте устройства памяти, поэтому отображение DIB-объекта обычно происходит медленнее, по сравнению с DDB-объектом.

Для преодоления ограничений, налагаемых особенностями DDB- и DIB-форматов, инженеры компании Microsoft разработали гибридный секционный DIB-растр. Он представляет собой комбинацию описанных выше двух типов и определяется структурой `DIBSECTION`.

```
typedef struct tagDIBSECTION {
    BITMAP              dsBm;
    BITMAPINFOHEADER    dsBmih;
    DWORD               dsBitfields[3];
    HANDLE              dshSection;
    DWORD               dsOffset;
}  DIBSECTION;
```

В отличие от оригинального формата DDB, биты секционного DIB-растра вполне доступны. В отличие от оригинального формата DIB, секционные DIB-растры могут выбираться в контексте устройства памяти. Эти два дополнительных аспекта особенно важны, когда требуется как эффективная манипуляция пикселями, так и эффективное отображение. Именно поэтому VCL-класс `TBitmap` базируется на секционном DIB-растре.

Класс `TBitmap`

Класс `TBitmap` — это VCL-инкапсуляция растрового объекта Windows. Этот класс выведен из абстрактного базового класса `TGraphic`, и он динамически адаптируется к использованию либо DDB-, либо секционного DIB-растра. Действия этого плана представлены внутренней VCL-функцией `CopyBitmap()`. Именно в этой функции происходит создание либо DDB-растра с помощью GDI-функций `CreateBitmap()` (для монохромных растров) и `CreateCompatibleBitmap()`, либо секционного DIB-растра с помощью GDI-функции `CreateDIBSection()`. Поскольку рассмотрение специфики VCL-функции `CopyBitmap()` выходит за рамки этого раздела, остановимся на тех свойствах и функциях-членах класса `TBitmap`, которые обеспечивают поддержку других типов изображений.

Класс TBitmap опирается на классы TBitmapCanvas и TBitmapImage. Класс TBitmapCanvas, потомок класса TCanvas, расширяет возможности своего родительского класса за счет инкапсуляции GDI-контекста устройства памяти. Класс TBitmapImage, потомок класса TSharedImage, обрабатывает ресурсы, проводя "инвентаризацию" и "списывание" DDB- или секционных DIB-растров. Списывание выполняется посредством GDI-функции DeleteObject().

Для поддержки отображения графических объектов класс TBitmapCanvas наследует от своего родительского класса функции-члены Draw(), StretchDraw() и CopyRect(). Функция TBitmapCanvas::CreateHandle() выполняет задачу выбора растра (и палитры при необходимости) в контексте устройства памяти. Следовательно, при использовании функции-члена Draw(), StretchDraw() или CopyRect() класс TCanvas может опираться исключительно на GDI-функцию StretchBlt() или TransparentStretchBlt().

Помимо функций-членов операций отображения, представленных классом TCanvas, в классе TBitmap предусмотрено свойство ScanLine. Это свойство использует внутреннюю функцию-член TBitmap::GetScanLine(), которая просто возвращает сдвинутый указатель на биты секционного DIB-растра. Выполняя преобразование из других форматов, вам придется получать доступ к этим битам, чтобы можно было непосредственно манипулировать пикселями изображения. В этом смысле свойство ScanLine значительно упрощает задачу манипуляции пикселями.

JPEG-изображения

JPEG (произносится "джейпег") — это не файловый формат, а протокол сжатия изображений, разработанный рабочей группой по стандартам цифровых видео- и мультипликационных изображений (Joint Photographic Experts Group). JPEG-изображения подвергаются сжатию с потерями, т.е. во время сжатия некоторая информация отбрасывается. Несмотря на то что распакованное изображение не идентично оригинальному, для большинства естественных изображений не наблюдается ухудшения качества или оно весьма несущественно. Более того, степенью сжатия можно управлять, что позволяет варьировать качество распакованных изображений.

Процесс JPEG-сжатия состоит из трех этапов: снижение пиксельной избыточности посредством дискретного косинусного преобразования (Discrete Cosine Transform — DCT), огрубления данных преобразования (DCT-коэффициенты) и уменьшения избыточности данных. Потеря данных (отбраковка) происходит на втором этапе (огрубление). Обычно именно здесь учитываются характеристики системы визуального восприятия человека (human visual system — HVS), несмотря на то, что не существует строгой спецификации для типа огрубления, которое должно быть выполнено. За подробностями об особенностях формата JPEG-изображений обращайтесь по адресу: http://www.jpeg.org/.

Библиотека VCL обеспечивает поддержку JPEG-изображений посредством класса TJPEGImage. Основные процедуры сжатия и распаковки выполняются с помощью библиотеки сжатия JPEG-изображений, созданной независимой рабочей группой JPEG (Independent JPEG Group — IJG). Подобно классу TBitmap, класс TJPEGImage является потомком класса TGraphic.

Класс TJPEGImage реализует функции-члены TPersistent::Assign() и AssignTo(), чтобы можно было без труда выполнять преобразования между экземплярами классов TJPEGImage и TBitmap. Более того, поскольку класс TJPEGImage предназначен для использования совместно с интерфейсом Windows GDI и другими компонентами библиотеки VCL, этот класс поддерживает внутреннее растровое представление изображения, чтобы сделать возможным про-

цесс отображения с помощью GDI. Сами же JPEG-данные поддерживаются посредством класса TJPEGData.

В классе TJPEGImage определен ряд свойств, предназначенных для управления JPEG-данными. Например, свойство CompressionQuality можно использовать для коррекции степени ухудшения, вносимого во время сжатия. Свойство Performance предназначено для манипуляции временем, необходимым для распаковки JPEG-данных. Свойство Scale определяет разрешение растра, получаемого в результате распаковки. Свойства ProgressiveDisplay и ProgressiveEncoding позволяют поддерживать постепенную распаковку, при которой можно просмотреть распакованную на данный момент часть изображения до того, как оно будет распаковано полностью. Эти свойства, а также свойство Smoothing, являются идеальными для ситуаций, в которых применяется схема постепенной передачи данных.

Поддержка операций работы с файлами, потоками и буфером обмена (Clipboard) обеспечивается посредством функций-членов LoadFromClipboardFormat(), LoadFromStream(), LoadFromFile(), SaveToClipboardFormat(), SaveToStream() и SaveToFile(). Все они унаследованы из класса TGraphic, а их использование не составляет большого труда. Обратите внимание, что для операций с буфером обмена класс TJPEGImage использует внутреннее представление растра.

GIF-изображения

Формат обмена графическими данными (Graphics Interchange Format — GIF, произносится "джиф") был создан в 1987 году корпорацией CompuServe. В отличие от JPEG-изображений, GIF-изображения сжимаются без потерь, т.е. во время сжатия данные не теряются. И поэтому распакованное изображение идентично оригинальному. GIF-изображения также поддерживают постепенное отображение (чередование), мульти-изображения (анимационный GIF) и прозрачность (в самой последней версии этого формата — GIF89a). Дополнительную информацию по GIF-изображениям можно получить по адресу: http://www.geocities.co.jp/SiliconValley/3453/gif_info/.

Многие разработчики неохотно поддерживают формат GIF, использование которого сопровождается лицензионными проблемами. На самом деле здесь стоит вопрос использования не формата как такового, а спецификации для использования алгоритма сжатия LZW (Lempel-Ziv-Welch). Эта модификация алгоритма сжатия Lempel-Ziv 78 (LZ78) запатентована корпорацией Unisys. CompuServe публично даровала лицензию на использование формата GIF, но корпорация Unisys требует, чтобы разработчики покупали у них лицензию.

В библиотеке VCL не предусмотрено встроенной поддержки GIF-формата. Для отображения GIF-изображений вам придется преобразовать (распакованные) данные в растровый формат. Несмотря на то что это можно сделать вручную с помощью свойства ScanLine, существует проблема распаковки и чтения данных из файла (или их записи). Существует ряд доступных библиотек сторонних производителей, которые позволяют справиться с этой задачей. Особый интерес вызывает свободно распространяемый компонент VCL TGIFImage из Anders Melanders, который можно найти по адресу: http://www.melander.dk/delphi/gifimage/.

PNG-изображения

Формат переносимой сетевой графики (Portable Network Graphics — PNG, произносится "пинг") был разработан для расширения GIF-формата и освобождения от его "патентных" проблем. Подобно своему предшественнику, PNG-изображение переносит процесс сжатия без потерь. В отличие от GIF-формата, PNG-изображения не опираются на алгоритм LZW.

Вместо него применяется вариация алгоритма сжатия Lempel-Ziv 77 (LZ77). Это именно тот алгоритм сжатия, который используется большинством приложений сжатия файлов.

Подобно GIF-формату, формат PNG позволяет использовать прозрачные пиксели. Однако при использовании канала Alpha PNG-изображения могут также содержать пиксели переменной прозрачности (alpha-смешивание). Более того, по сравнению с GIF-форматом, PNG-изображения не ограничены 256 цветами. Чтобы компенсировать изменения монитора, PNG-спецификация позволяет использовать закодированную гамма-информацию. Предусмотрена также поддержка постепенного отображения, выполняемая посредством схемы двумерного чередования. К сожалению, PNG-формат не позволяет использовать мульти-изображения (анимацию).

Чтобы обеспечить отображение PNG-изображений, вам понадобятся средства преобразования PNG-формата в DIB-формат. Как всегда, это можно выполнить вручную; в этом случае понадобятся самые последние спецификации формата PNG. Их можно найти по адресу: `http://www.libpng.org/pub/png/spec/PNG-Contents.html`. Как и со многими другими форматами изображений, существует ряд библиотек сторонних производителей, которые могут выполнить это преобразование за вас. Например, свободно распространяемая библиотека преобразований `PNGDIB` Джейсона Саммерса (Jason Summers), которую можно отыскать по адресу: `http://home.mieweb.com/jason/imaging/pngdib/`, содержит функцию `read_png_to_dib()`, способную прочитать файл PNG-изображения и выдать в качестве результата DIB-растр (`BITMAPINFO`, цветовую таблицу и биты). Эта библиотека также предоставляет в ваше распоряжение функцию `write_dib_to_png()` для записи PNG-файла из DIB-растра.

Используя эту библиотеку для инициализации объекта `TBitmap` в соответствии с информацией, содержащейся в PNG-файле, сначала нужно вызвать функцию `read_png_to_dib()` (чтобы создать DIB-растр из PNG-изображения), а затем использовать GDI-функцию `SetDIBits()` (или свойство `ScanLine`) для заполнения объекта `TBitmap`. Этот процесс демонстрируется в листинге 25.1.

Листинг 25.1. Преобразование формата PNG-изображения в объект класса `TBitmap`

```
if (OpenDialog1->Execute())
{
   TCHAR filename[MAX_PATH];
   lstrcpyn(filename, OpenDialog1->FileName.c_str(), MAX_PATH);

   // Объявляем и очищаем структуру PNGD_P2DINFO.
   PNGD_P2DINFO png2dib;
   memset(&png2dib, 0, sizeof(PNGD_P2DINFO));

   // Инициализируем размер структуры и имя файла.
   png2dib.structsize = sizeof(PNGD_P2DINFO);
   png2dib.pngfn = filename;

   // Выполняем преобразование из PNG-формата в DIB-растр.
   if (read_png_to_dib(&png2dib) == PNGD_E_SUCCESS)
   {
      Graphics::TBitmap* Bitmap = Image1->Picture->Bitmap;
      Bitmap->Width = png2dib.lpdib->biWidth;
      Bitmap->Height = png2dib.lpdib->biHeight;

      HBITMAP hBmp = Bitmap->ReleaseHandle();
```

```
      HDC hDC = Canvas->Handle;
      try
      {
         //
         // СДЕЛАТЬ: добавить поддержку палитры...
         //

         // Выполняем преобразование из DIB-растра в TBitmap.
         SetDIBits(
            hDC, hBmp, 0,
            png2dib.lpdib->biHeight, png2dib.bits,
            reinterpret_cast<LPBITMAPINFO>(png2dib.lpdib),
            DIB_RGB_COLORS
            );
      }
      catch (...)
      {
         Bitmap->Handle = hBmp;
         GlobalFree(png2dib.lpdib);
      }
      Bitmap->Handle = hBmp;
      GlobalFree(png2dib.lpdib);
   }
}
```

Аналогично, чтобы создать PNG-файл из информации, содержащейся в объекте класса `TBitmap`, сначала создайте DIB-растр из данных типа `TBitmap` с помощью VCL-функций `GetDIBSizes()` и `GetDIB()` (из файла `graphics.pas`), а затем используйте функцию `write_dib_to_png()` для записи PNG-файла. Этот процесс продемонстрирован в листинге 25.2.

Листинг 25.2. Преобразование объекта класса `TBitmap` в формат PNG

```
if (SaveDialog1->Execute())
{
   TCHAR filename[MAX_PATH];
   lstrcpyn(filename, SaveDialog1->FileName.c_str(), MAX_PATH);

   BITMAPINFO bmi;
   Graphics::TBitmap* Bitmap = Image1->Picture->Bitmap;

   //
   // Определяем размер DIB-информации: (BITMAPINFOHEADER + цветовая таблица)
   // плюс размер памяти для битов (пикселей).
   //
   unsigned int info_size = 0, bits_size = 0;
   GetDIBSizes(Bitmap->Handle, info_size, bits_size);

   // Выделяем память для битов.
   unsigned char *bits = new unsigned char[bits_size];
```

```
   try
   {
      // Получаем: BITMAPINFOHEADER + цветовая таблица и биты.
      if (GetDIB(Bitmap->Handle, Bitmap->Palette, &bmi, bits))
      {
         // Объявляем и очищаем структуру PNGD_D2PINFO.
         PNGD_D2PINFO dib2png;
         memset(&dib2png, 0, sizeof(PNGD_D2PINFO));

         // Инициализируем эту структуру.
         dib2png.structsize = sizeof(PNGD_D2PINFO);
         dib2png.flags = PNGD_INTERLACED;
         dib2png.pngfn = filename;
         dib2png.lpdib = &bmi.bmiHeader;
         dib2png.lpbits = bits;

         // Преобразуем DIB-растр в PNG-формат, затем сохраняем в файле.
         if (write_dib_to_png(&dib2png) != PNGD_E_SUCCESS)
         {
            throw EInvalidGraphic("Ошибка при сохранении PNG!");
         }
      }
   }
   catch (...)
   {
      delete [] bits;
   }
   delete [] bits;
}
```

Использование библиотеки PNGDIB демонстрируется в проекте Proj_PNGDIBDemo.cpp, который находится на прилагаемом к книге компакт-диске (в папке PNGDemo).

Обработка изображений

Под обработкой изображений обычно понимают манипуляцию и анализ "изобразительной" информации цифровыми компьютерами. Диапазон приложений обработки изображений довольно широк: от небольших цифровых фотоаппаратов до современных военных систем.

Изображение хранится в компьютере в виде двумерного массива чисел, что упрощает манипуляцию ими. Каждый элемент изображения называется *пикселем*. Количество пикселей, используемое для хранения изображения, называется *разрешением*. Чем больше это количество пикселей, тем выше разрешение. В этом разделе мы поработаем с несколькими изображениями — с разрешением 256×256 и 512×512.

Количество байт, необходимых для представления пикселя, зависит от количества оттенков серого цвета, которое может отображать пиксель. В случае использования черно-белого изображения достаточно иметь один бит на пиксель. Для ахроматических (в оттенках серого) изображений принято считать, что система визуального восприятия человека не может различать более 256 оттенков. Следовательно, одного байта на пиксель будет вполне достаточно. В случае настоящих цветных изображений каждый пиксель представляется триплетом,

состоящим из красного, зеленого и синего цветов, причем каждая составляющая выражается одним байтом, т.е. на пиксель приходится три байта. Простое вычисление показывает, что для хранения цветного изображения с разрешением 512×512 вам понадобится 512×512×3 байт = 768 Кбайт информации. Для хранения и манипуляции изображениями требуется большой объем памяти, поэтому обработку изображений до недавнего времени выполняли только самые "крутые" системы. А ведь компьютер Apple II (в то время, когда я начинал программировать) имел только 48 Кбайт доступного ОЗУ!

Данные изображения редко хранятся в необработанном виде. Для различных приложений можно использовать множество различных форматов изображений. Встроенным форматом изображений в среде Windows является Windows Bitmap (BMP), который и будет использован в этой главе. Другими популярными форматами являются TIFF и PCX.

Мы не планируем здесь углубляться в подробные математические описания методов, используемых в обработке изображений. Но на основных методах остановимся и рассмотрим примеры, созданные с помощью C++Builder. Программа обработки изображений, представленная в этом разделе, входит в проект `IPro.bpr`, который находится в папке `ImageProcessing` на прилагаемом к книге компакт-диске. На рис. 25.3 показано, как это приложение обработки изображений выглядит в действии, отображая файл `peppers.bmp`.

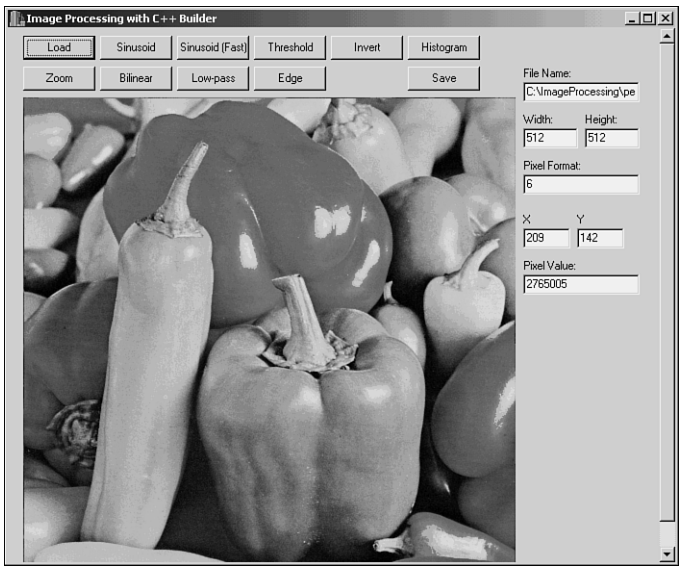


Рис. 25.3. Простое приложение обработки изображений

Отображение и получение графической информации

Первое, что хочется сделать с любым изображением, — отобразить его и "выудить" из его файла такую информацию, как разрешение и глубина цвета. Проще всего это сделать с помощью компонента `TImage`, который находится во вкладке Additional (Дополнительно) палитры компонентов. Поместив его в форму, целесообразно установить для его свойства `AutoSize` значение `true`, чтобы этот компонент автоматически настраивался под различные размеры изображений. Программа, приведенная в листинге 25.3, по щелчку на соответствующей кнопке загружает в компонент изображение, выбранное в диалоговом окне Open.

Затем она отображает в нескольких полях редактирования основную информацию об этом изображении. В данном проекте для имен компонентов используются значения, действующие по умолчанию.

Листинг 25.3. Загрузка изображения и получение информации о нем

```
void __fastcall TForm1::Button1Click(TObject *Sender)
{
    OpenDialog1->Filter = "Bmp files (*.bmp)|*.BMP";
    if (OpenDialog1->Execute())
    {
        Image1->Picture->Bitmap->LoadFromFile(OpenDialog1->FileName);
        Edit1->Text = OpenDialog1->FileName;
          Edit2->Text = Image1->Width;
        Edit3->Text = Image1->Height;
        Edit4->Text = Image1->Picture->Bitmap->PixelFormat;
    }
}
```

Первый оператор отображает диалоговое окно Open в виде объекта `OpenDialog1` типа `TOpenDialog`, с использованием фильтра для BMP-файлов. Если это диалоговое окно возвращает значение `true`, значит, пользователь выбрал файл, который затем загружается в "растровый" компонент `Image1`. Для получения информации нужно получить доступ к свойствам компонента `Image1`. В частности, `PixelFormat` — это свойство перечислимого типа, определенного следующим образом в заголовочном файле `graphics.hpp` C++Builder.

```
enum TPixelFormat { pfDevice, pf1bit, pf4bit, pf8bit, pf15bit,
                            pf16bit, pf24bit, pf32bit, pfCustom} ;
```

Следовательно, 24-разрядному цветному изображению будет соответствовать числовое значение `6`, которое содержится в свойстве `PixelFormat`. Для 8-разрядного изображения в оттенках серого свойство `PixelFormat` имеет значение `3`. Следует иметь в виду, что, хотя функция `Load()` может обрабатывать BMP-файлы всех размеров и значений глубины цвета, большинство других функций работают в предположении (ради простоты), что используется 8-разрядное в оттенках серого изображение размером 256×256.

На рис. 25.3 показана полученная нами информация об изображении, которую мы поместили в верхние четыре поля редактирования справа от изображения.

Доступ к значениям отдельных пикселей с помощью свойства `TCanvas->Pixels`

Прежде чем выполнять какие-либо операции по обработке изображений, необходимо иметь доступ к значениям отдельных пикселей изображения. Проще всего такой доступ получить с помощью свойства `Pixels` компонента `TCanvas`. Это свойство можно использовать как двумерный массив с координатами `(x,y)`, измеряемыми начиная от верхнего левого угла изображения. В листинге 25.4 демонстрируется, как получить доступ к позиции мыши и значению пикселей, а также показать пользователю перемещение указателя мыши по изображению. Эту информацию можно увидеть в нижних трех полях редактирования (см. рис. 25.3).

Листинг 25.4. Получение значения пикселя при перемещении указателя мыши

```
void __fastcall TForm1::Image1MouseMove(TObject *Sender,
                                        TShiftState Shift, int X, int Y)
{
    Edit5->Text = X;
    Edit6->Text = Y;
    Edit7->Text = Image1->Canvas->Pixels[X][Y];
}
```

Текущая координата мыши передается в обработчик события в качестве параметров `X` и `Y`. Значение пикселя имеет тип `TColor`, а его интерпретация зависит от значение свойства `PixelFormat`. Для изображений в оттенках серого каждый оттенок содержится в младшем байте значения пикселя. Для 24-разрядных цветных изображений младшие три байта представляют RGB-интенсивности для синего, зеленого и красного, соответственно. Значение `$00FF0000` представляет синий цвет, `$0000FF00` — зеленый, а `$000000FF` — красный.

Вы можете также установить значение пикселя с помощью свойства `TCanvas->Pixels[][]`. Например, после добавления следующей строки в обработчик событий `Image1MouseMove()` будет нарисована линия, повторяющая движение указателя мыши.

```
Image1->Canvas->Pixels[X][Y] = clWhite;
```

Свойство `Canvas->Pixels[X][Y]` очень просто использовать, но работает оно чрезвычайно медленно. Поэтому применять его для доступа к значениям пикселей стоит лишь изредка. Самый же эффективный способ получения доступа к данным изображения — с помощью свойства `ScanLine`, о котором речь пойдет ниже, в разделе "Быстрый доступ к значению пикселя с помощью свойства `ScanLine`".

Создание изображений

Очень часто специалистам по обработке изображений приходится создавать тестовые изображения для последующей работы с ними, поскольку будут заведомо известны их статистические данные. Мы создадим тестовое изображение, которое обеспечивает некоторое проникновение в свойства системы визуального представления человека. Это изображение представляет собой синусоидальные колебания оттенков серого в горизонтальном направлении. Частота колебаний, именуемая пространственной частотой, возрастает с увеличением значения `x`. Амплитуда синусоидальной волны линейно уменьшается с увеличением значения `y`. Программа создания этого изображения представлена в листинге 25.5. Следует отметить, что следующий код использует стандартную математическую функцию `sqrt()`, поэтому в модуль необходимо включить заголовочный файл `math.h`. При необходимости вы можете обратиться к полному проекту (`IPro.bpr`), включенному в состав прилагаемого к книге компакт-диска.

Листинг 25.5. Создание синусоидального изображения с помощью свойства `TCanvas->Pixels`

```
void __fastcall TForm1::Button2Click(TObject *Sender)
{
    int x, y;
    int Amplitude, Gray;
    float Period;
    TColor RGB;
```

```
    Image1->Picture->Bitmap = new Graphics::TBitmap;
    Image1->Picture->Bitmap->PixelFormat = pf24bit;
    Image1->Picture->Bitmap->Width = 256;
    Image1->Picture->Bitmap->Height = 256;
    for (y=0; y<=255; y++)
        for (x=0; x<=255; x++)
        {
            Amplitude = 64*(255-y)/255;
            Period = 100*sqrt(1/(1+(exp(0.013*x)*exp(0.027*x)/400)));
            Gray = Amplitude*sin(2*M_PI/Period*x)+128;
            RGB = (Gray << 16) | (Gray << 8) | Gray;
            Image1->Canvas->Pixels[x][y] = RGB;
        }
}
```

Сначала создается объект класса `TBitmap`, на котором мы будем рисовать. Затем устанавливаются такие свойства изображения, как `PixelFormat`, `Width` и `Height`. Внутри вложенного цикла `for` вычисляется значение каждого пикселя. Математическое обоснование выходит за рамки этой книги, поэтому переходим к рисованию, которое заключается в установке значений свойства `Image1->Canvas->Pixels`. В результате создается изображение, показанное на рис. 25.4.

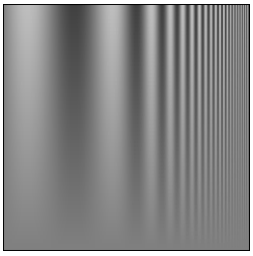


Рис. 25.4. Изменение синусоиды

На заметку

Из рис. 25.4 видно, что чувствительность визуальной системы к контрасту меняется вместе с пространственной частотой, достигая максимума на средних частотах и падая на очень низких или очень высоких частотах. Этот вид информации о нашей визуальной системе весьма полезен в приложениях, связанных с передачей и сжатием изображений, поскольку он помогает определить, каким образом выделить диапазон рабочих частот или пространство для хранения информации, содержащейся в изображении. Менее важной информации (которую все равно нельзя разглядеть достаточно хорошо) следует отвести более узкий диапазон рабочих частот или меньший объем памяти.

Наконец, нам нужно сохранить изображение для последующей работы с ним. Это можно сделать с помощью функции `SaveToFile()` по щелчку на кнопке, что продемонстрировано в следующем фрагменте программы.

```
void __fastcall TForm1::Button5Click(TObject *Sender)
{
        Image1->Picture->Bitmap->SaveToFile("test.bmp");
}
```

Вместо “жесткого” задания имени файла `test.bmp` можно было бы разрешить пользователю самому указать имя файла с помощью компонента `TSaveDialog`.

Быстрый доступ к значению пикселя с помощью свойства ScanLine

Даже при наличии компьютера Pentium III 500MHz PC, предыдущий пример для создания изображения размером 256×256 потребует секунду или две. Дело в том, что использование свойства TCanvas->Pixels требует больших расходов системных ресурсов. Поэтому лучше выполнить операции по обработке изображения в двумерном массиве, а затем скопировать результат в компонент TBitmap, используя свойство ScanLine. Листинг 25.6 — еще один вариант последнего примера, в котором вместо свойства Pixels используется свойство ScanLine.

Листинг 25.6. Создание синусоидального изображения с помощью свойства ScanLine

```
void __fastcall TForm1::Button3Click(TObject *Sender)
{
    int x, y;
    int Amplitude;
    float Period;
    BYTE ImageData[256][256], *LinePtr;

    Image1->Picture->Bitmap = new Graphics::TBitmap;
    Image1->Picture->Bitmap->PixelFormat = pf24bit;
    Image1->Picture->Bitmap->Width = 256;
    Image1->Picture->Bitmap->Height = 256;
    for (y=0; y<=255; y++)
        for (x=0; x<=255; x++)
        {
            Amplitude = 64*(255-y)/255;
            Period = 100*sqrt(1/(1+(exp(0.013*x)*exp(0.027*x)/400)));
            ImageData[x][y] = Amplitude*sin(2*M_PI/Period*x)+128;
        }
    // Копируем данные изображения в объект Bitmap.
    for (y=0; y<=255; y++)
    {
        LinePtr = (BYTE *) Image1->Picture->Bitmap->ScanLine[y];
        for (x=0; x<=255; x++)
        {
            LinePtr[x*3]   = ImageData[x][y];  // Red (красный)
            LinePtr[x*3+1] = ImageData[x][y];  // Green (зеленый)
            LinePtr[x*3+2] = ImageData[x][y];  // Blue (синий)
        }
    }
    Image1->Refresh();
}
```

Двумерный массив байт ImageData используется для хранения вычисленных данных изображения. После вычисления всего изображения данные построчно копируются в растровый объект. Переменная LinePtr используется для хранения указателя на первый пиксель линии, возвращаемый свойством ScanLine компонента TCanvas. Каждое значение пикселя должно быть скопировано три раза для трех цветовых компонентов. При использовании этого варианта изображение после запуска программы появляется почти моментально.

Точечные операции: „пороговая“ обработка и инвертирование цветных и ахроматических изображений

Точечные операции представляют собой процессы, которые попиксельно изменяют градации серого в изображении. Точечные операции могут зависеть от позиции пикселя или его значения. В последнем случае для операций иногда употребляют термин *преобразование оттенков серого*. Преобразование в зависимости от позиции обычно используется для коррекции дефектов фотоаппарата или неровного освещения. Преобразование оттенков серого, как правило, применяют для улучшения изображений низкого качества. Изображение (файл `splash.bmp`, находящийся на компакт-диске, прилагаемом к книге), используемое здесь для демонстрации точечных операций, показано на рис. 25.5.

“Пороговая” обработка (thresholding) — самая простая форма преобразования оттенков серого. Значение пикселя преобразуется в соответствии с условиями, приведенными на рис. 25.6. Значение серого оттенка устанавливается равным `0` (нуль — черный цвет), если его исходное значение меньше заданного порога (порогового значения), или устанавливается равным числу `255` (белый цвет), если исходное значение больше или равно заданному порогу. Такая обработка пороговыми значениями просто превращает изображение, созданное в оттенках серого, в черно-белое (без каких-либо оттенков серого).

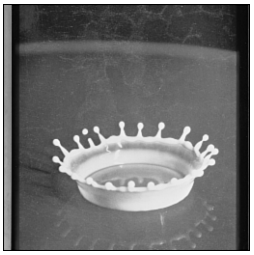


Рис. 25.5. Исходное изображение для точечных операций

$$\begin{cases} = 0 & < threshold \\ = 255 & \geq threshold \end{cases}$$

Рис. 25.6. Обработка изображения пороговыми значениями

В качестве порога может выступать фиксированное или динамически определяемое значение. Типичным примером является преобразование изображения, созданного в оттенках серого, в черно-белое. Это называется *преобразованием в двоичную форму* (binarization).

В листинге 25.7 каждое значение пикселя сравнивается с пороговым значением 128. Если оно оказывается выше этого значения, пиксель “окрашивается” в белый цвет; в противном случае — в черный. Результат применения пороговой обработки к изображению, показанному на рис. 25.5, демонстрируется на рис. 25.7.

Листинг 25.7. Обработка изображения пороговыми значениями

```
void __fastcall TForm1::Button11Click(TObject *Sender)
{
   int x, y;
   BYTE* LinePtr;

   // Обработка изображения пороговыми значениями.
```

```
   for (y=0; y<=255; y++)
   {
        LinePtr=(BYTE*)Image1->Picture->Bitmap->ScanLine[y];
        for (x=0; x<=255; x++)
            if (LinePtr[x] > 128)
                LinePtr[x] = 255;
            else
                LinePtr[x] = 0;
   }
   Image1->Refresh ();
}
```

Определение порогового значения может оказаться трудной задачей, часто предполагающей наличие предварительной информации об изображении. Существует и альтернативный вариант, когда порог можно вычислить, опираясь на статистические данные изображения (например, среднее значение или квадратичное отклонение), но его реализация выходит за рамки этой книги.

Рассмотрим еще один пример применения точечных операций — инвертирование ахроматических изображений, с помощью которого можно получить негатив изображения. Для этого достаточно выполнить операцию вычитания, как показано в листинге 25.8.

Листинг 25.8. Инвертирование изображения

```
void __fastcall TForm1::Button4Click(TObject *Sender)
{
    int x, y;
    BYTE* LinePtr;

    // Инвертирование изображения.
    for (y=0; y<=255; y++)
    {
          LinePtr=(BYTE*)Image1->Picture->Bitmap->ScanLine[y];
          for (x=0; x<=255; x++)
              LinePtr[x] = 255 - LinePtr[x];
    }
   Image1->Refresh ();
}
```

Результат инвертирования исходного изображения (см. рис. 25.5) показан на рис. 25.8, который выглядит, как негатив фотографии.

Рис. 25.7. Результат обработки пороговыми значениями

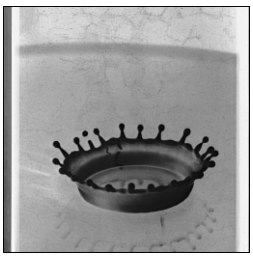


Рис. 25.8. Результат инвертирования изображения

Глобальная операция: выравнивание гистограмм

Под глобальной операцией подразумевается обработка на основе глобальных свойств изображения. Слово *глобальный* означает, что эти свойства должны быть выведены из целого изображения. Типичным примером является выравнивание гистограмм. В зависимости от освещения изображение может неполностью использовать динамический диапазон. В этом случае изображение будет отличаться низкой контрастностью. Пример такого изображения показан на рис. 25.9 (его файл `couple.bmp` находится на прилагаемом компакт-диске к книге).

Гистограмма этого изображения показана на рис. 25.10. Гистограмма — это построение графика на основе чисел, которые соответствуют каждой градации серого цвета в изображении. Значения серых оттенков от 0 до 255 отображаются в горизонтальном направлении слева направо. Чем больше раз встречается данный оттенок, тем выше линия в соответствующей позиции гистограммы. Если рассмотреть гистограмму пиксельных значений изображения, можно увидеть, что она смещена влево, а это значит, что большинство пикселей довольно темны.

Рис. 25.9. Малоконтрастное изображение

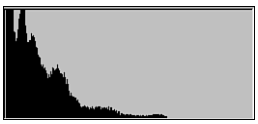


Рис. 25.10. Гистограмма малоконтрастного изображения

Первый этап в процессе выравнивания гистограммы — получение гистограммы полутонового распределения изображения. Поэтому мы рассмотрим фрагмент функции `TForm1::Button10Click()`, используемой в нашем примере приложения.

В листинге 25.9 переменная `Histogram` объявляется как массив для хранения 256 чисел (256 — это количество различных оттенков серого для 8-разрядного изображения). Переменная `Histogram` первоначально инициализируется значением `0` (нуль); затем для получения значения каждого пикселя в изображении увеличивается соответствующий элемент гистограммы (счетчик для каждого значения серого оттенка).

Листинг 25.9. Вычисление элементов гистограммы

```
// Инициализируем гистограмму.
    for (i=0; i<=255; i++)
        Histogram[i] = 0;
    // Строим гистограмму.
    for (y=0; y<=255; y++)
        for (x=0; x<=255; x++)
            Histogram[ImageData[x][y]]++;
```

После построения гистограммы создается таблица для преобразования исходного значения серого оттенка. Эта таблица преобразования строится таким образом, чтобы распределение значений пикселя после трансформации стало почти равномерным.

$$v = \frac{255}{256 \times 256} \sum_{i=0}^{u} Histgram(i)$$

Рис. 25.11. Преобразование значения пикселя

Преобразование значения пикселя `u` в значение `v` определяется выражением, приведенным на рис. 25.11.

В листинге 25.10 показано, как выражение, приведенное на рис. 25.11, можно записать в виде программы.

Листинг 25.10. Вычисление преобразования

```
// Вычисление преобразования.
    for (i=0; i<=255; i++)
    {
        sum = 0;
        for (j=0; j<=i; j++)
            sum += Histogram[j];
        Transform[i] = INT(255.0*sum/(256*256));
    }
```

Переменная `Transform` представляет собой массив, состоящий из 256 элементов, в которых хранятся преобразованные значения для каждого возможного значения пикселя. Наконец, каждое значение пикселя преобразуется в новое значение, как показано в листинге 25.11.

Листинг 25.11. Преобразование изображения

```
// Преобразование изображения.
    for (y=0; y<=255; y++)
        for (x=0; x<=255; x++)
            ImageData[x][y] = Transform[ImageData[x][y]];
```

Результат применения гистограммного выравнивания к нашему изображению показан на рис. 25.12. Соответствующая ему гистограмма показана на рис. 25.13.

Рис. 25.12. Изображение после гистограммного выравнивания

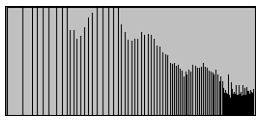


Рис. 25.13. Гистограмма после выравнивания

Теперь контраст изображения стал заметно лучше. Кроме того, гистограмма изображения имеет более равномерное распределение, чем раньше, демонстрируя улучшенное использование доступного динамического диапазона.

Геометрическое преобразование: масштабирование

Геометрические преобразования могут быть линейными и нелинейными. К *линейным геометрическим преобразованиям* относятся параллельный перенос (сдвиг), вращение, масштабирование или их комбинации. (Они также называются *аффинными преобразованиями*.) К *нелинейным геометрическим преобразованиям* относятся изгиб и искажение изображения. Они также называются деформирующими преобразованиями. В качестве примера геометрического преобразования будет реализовано масштабирование. Исходное изображение (его файл, `peppers8bit.bmp`, находится на прилагаемом к книге компакт-диске), показано на рис. 25.14.

Уравнения для вычисления новой позиции пикселя при масштабировании представлены на рис. 25.15.

Здесь `(x',y')` — координаты новой позиции для пикселя с координатами `(x,y)`, `a` — коэффициент масштабирования, а (x_0,y_0) — центр масштабирования. Однако поскольку вычисленные новые позиции зачастую не являются целыми числами, обычной практикой является применение приведенных выше уравнений в обратном порядке. Для каждого пикселя в новом изображении с помощью уравнений, представленных на рис. 25.16, вычисляется исходная позиция пикселя.

Рис. 25.14. Исходное изображение, используемое для последующего масштабирования

$$x' = a(x - x_0) + x_0$$
$$y' = a(y - y_0) + y_0$$

Рис. 25.15. Прямое вычисление позиции пикселя

$$x = \frac{x' + (a-1)x_0}{a}$$
$$y = \frac{y' + (a-1)y_0}{a}$$

Рис. 25.16. Обратное вычисление позиции пикселя

На рис. 25.17 видно, что вычисленная позиция в общем случае будет попадать на "нецелую" позицию, но на этот раз для получения значения пикселя можно использовать соседние пиксели.

Проще всего, конечно, взять значение ближайшего соседнего пикселя. Этот метод демонстрируется в листинге 25.12, а его полный код содержится в функции `TForm1::Button6Click()` примера приложения.

Листинг 25.12. Масштабирование изображения с помощью метода ближайшего соседнего пикселя

```
// Вычисляем результирующее изображение.
    for (y=0; y<=255; y++)
        for (x=0; x<=255; x++)
        {
            // Ближайший соседний пиксель.
            i = INT((x+(zoom-1)*x0) / zoom ); // x0 = 127,
                                             // y0 = 127
```

```
        j = INT((y+(zoom-1)*y0) / zoom ); // zoom = 4
        Output[x][y] = ImageData[i][j];   // Сохраняем вычисленное значение.
    }
```

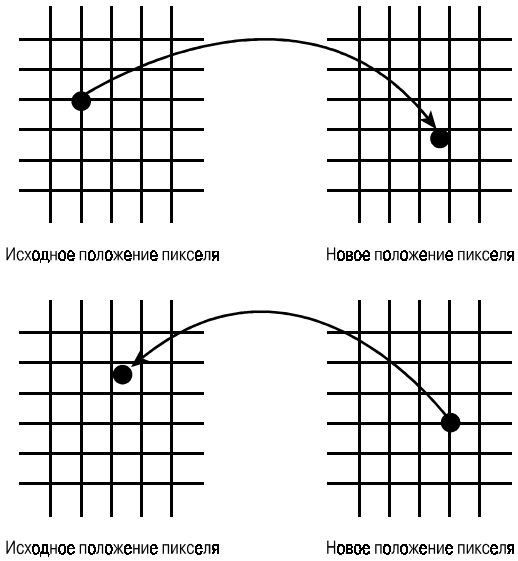


Рис. 25.17. Прямое и обратное вычисление позиции пикселя

Следует отметить, что вычисленное заново изображение нужно сохранять в другом массиве. Распространенной ошибкой является запись вычисленного значения в исходный массив изображения, что ведет к непредсказуемым результатам.

Глядя на результирующее изображение, можно говорить о том, что эффект “блочности” весьма заметен. Результат можно улучшить, используя вместо метода ближайшего соседа билинейную интерполяцию. В билинейной интерполяции используются четыре ближайших пикселя (относительно вычисленной позиции) для получения взвешенного среднего с помощью формулы, визуальная интерпретация которой представлена на рис. 25.18.

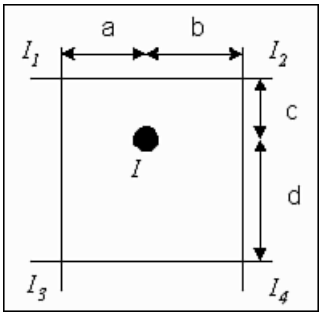


Рис. 25.18. Билинейная интерполяция значения пикселя

Код реализации билинейной интерполяции приведен в листинге 25.13, содержащем функцию `Bilinear()`. Эта функция представляет собой один из фрагментов функции `TForm1::Button7Click()`.

Листинг 25.13. Масштабирование изображения с помощью метода билинейной интерполяции

```
int Bilinear(BYTE Image[][256], float i, float j)
{
    int x1, y1, x2, y2;
    float Gray;

    x1 = (int)floor(i);
    x2 = x1+1;
    y1 = (int)floor(j);
    y2 = y1+1;
    Gray = (y2-j)*(x2-i)*Image[x1][y1] +
           (y2-j)*(i-x1)*Image[x2][y1] +
           (j-y1)*(x2-i)*Image[x1][y2] +
           (j-y1)*(i-x1)*Image[x2][y2];
    return INT(Gray);
}

void __fastcall TForm1::Button7Click(TObject *Sender)
{
    // Код инициализации не показан.

    // Вычисляем результирующее изображение.
    for (y=0; y<=255; y++)
        for (x=0; x<=255; x++)
        {
            i = (x+(zoom-1)*x0) / zoom; // Вычисляем исходную позицию.
            j = (y+(zoom-1)*y0) / zoom;
            Output[x][y] = Bilinear(ImageData, i, j);
        }

    // Копирование обратно в изображение не показано.
}
```

Рис. 25.19. Изображение, увеличенное в четыре раза с помощью метода ближайшего соседнего пикселя

Рис. 25.20. Изображение, увеличенное в четыре раза с помощью метода билинейной интерполяции

Сравнив результирующие изображения, показанные на рис. 25.19 и рис. 25.20, можно сделать вывод о том, что изображение, полученное с помощью метода билинейной интерполяции, отличается большей гладкостью и вообще имеет более привлекательный вид. Чтобы получить еще более впечатляющие результаты, можно использовать более высокий порядок интерполяции (например, кубическую), что, естественно, усложняет математическую базу этого метода.

Пространственная операция: сглаживание и выделение контуров изображения

Пространственная операция относится к таким методам обработки изображений, которые воздействуют не на один пиксель, а на целую группу пикселей. В эту категорию попадает широкий диапазон методов, да и цели применения пространственных операций могут быть весьма различными.

В пространственных операциях для определения интересующей пользователя области и значения каждого пикселя используется маска. Опираясь на маску, пикселю, расположенному в центре этой области, присваивается заново вычисленное значение. Из практических соображений маска обычно берется квадратной, с размерами 3×3 и 5×5. Однако размер маски может даже быть 17×17, как в случае функции "Мексиканская шляпа" (Mexican-Hat function), свойства которой подобны сетчатке человеческого глаза и поэтому часто используется в компьютерных видеосистемах.

Самая простая пространственная фильтрация реализуется с помощью фильтра сглаживания, который имеет маску 3×3, имеющую следующий вид.

```
1/9 1/9 1/9
1/9 1/9 1/9
1/9 1/9 1/9
```

Значение каждого пикселя заменяется средним значением, вычисленным из расположенных по соседству 3×3 пикселей. Этот метод называется низкочастотным фильтром, поскольку создает эффект затухания высокочастотных компонентов изображения. Этот метод представлен в функции `Lowpass()`, которая вызывается из функции `TForm1::Button8Click()` (листинг 25.14).

Листинг 25.14. Сглаживание изображения с помощью маски 3×3

```
int Lowpass(BYTE Image[][256], int x, int y)
{
    int Mask[3][3] = { { 1, 1, 1,} , { 1, 1, 1} , { 1, 1, 1} } ;
    int i, j, sum=0;

    for (j=-1; j<=1; j++)
        for (i=-1; i<=1; i++)
            sum += Image[x+i][y+j]*Mask[i+1][j+1];
    return INT(sum/9.0);
}

void __fastcall TForm1::Button8Click(TObject *Sender)
{
    // Код инициализации не показан.
```

```
    // Вычисляем низкочастотное изображение.
    for (y=1; y<=254; y++)
        for (x=1; x<=254; x++)
            Output[x][y] = Lowpass(ImageData, x, y);

    // Копирование обратно в изображение не показано.
}
```

В нашем изображении эффект выравнивания проявляется очень заметно. Результат применения маски к изображению, показанному на рис. 25.14, четко виден на рис. 25.21. При использовании большей маски эффект сглаживания усиливается.

Для высокочастотной фильтрации обычно используется маска

```
-1/9 -1/9 -1/9
-1/9  1   -1/9
-1/9 -1/9 -1/9
```

Отрицательные значения в маске говорят о том, что, если значение пикселя превышает значения соседних пикселей, оно должно быть увеличено. Выделение контуров изображения — это форма высокочастотной фильтрации, поскольку края (контуры) в изображении ассоциируются с высокими пространственными частотами, т.е. областями с быстроменяющимся содержимым. Выделение контуров — важный метод предварительной обработки, часто применяемый в компьютерных видеосистемах, поскольку он облегчает определение границ объекта, а это является первым шагом к распознаванию объектов. Результат применения предыдущей маски к изображению на рис. 25.14 показан на рис. 25.22.

Рис. 25.21. Изображение, подвергнутое низкочастотной фильтрации с применением маски 3×3

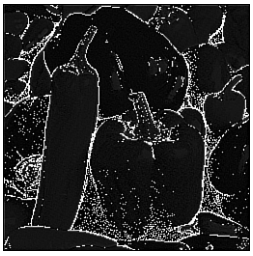


Рис. 25.22. Изображение, подвергнутое высокочастотной фильтрации с маской 3×3

Следует обратить ваше внимание на одно осложнение, возникающее при реализации этого метода. Когда маска применяется к пикселям в краевой области изображения, она выходит за его пределы. Во избежание этой проблемы проще всего оставить краевые пиксели в исходном виде, а затем скопировать в них значения следующих доступных пикселей после их обработки. Для улучшения результатов можно применить интерполяцию.

Аудиофайлы, видеофайлы и музыка на компакт-дисках

Мультимедийная система Windows предоставляет стандартные средства для управления мультимедийными устройствами. Эта система просто осуществляет связь между приложением и конкретным драйвером устройства. Интерфейс управления средствами аудиовизуальной информации (Media Control Interface — MCI) еще в большей степени увеличивает гибкость, обеспечивая средства, с помощью которых приложения могут взаимодействовать со всеми поддерживаемыми аудио- и видеоустройствами. С точки зрения разработчика, специфика конкретного устройства вообще не имеет значения; а зачастую его не волнует даже тип устройства.

В этом разделе мы сначала рассмотрим использование интерфейса MCI для создания аудио- и видеоприложений общего назначения. (Как вы скоро убедитесь, интерфейс MCI является, возможно, самым простым мультимедийным интерфейсом). Затем мы остановимся на использовании интерфейса, применяемого для восстановления волновой формы сигнала из дискретизированной (waveform-audio interface). Наконец, мы коснемся проблемы аудиопотоков и познакомимся с особенностями чтения и записи музыкальных (waveform-audio) файлов.

Интерфейс управления средствами аудиовизуальной информации (MCI)

Подобно тому как интерфейс GDI предлагает программисту общие средства коммуникации с графическими устройствами, интерфейс Windows MCI позволяет программировать мультимедийные устройства, причем в аппаратно-независимой форме. До существования интерфейса MCI разработчики были вынуждены писать программы в расчете на конкретные устройства. Часто этот процесс предполагал использование процедур, ориентированных на драйвер определенного устройства. Нетрудно представить в этом случае существование схемы, в которой потребитель был ограничен определенным диапазоном типов устройств. Например, во многие из первых DOS-ориентированных игр было заложено требование, чтобы звуковая карта была совместима со стандартом Sound Blaster.

Использование командных сообщений и строковых констант

Приложения взаимодействуют с интерфейсом MCI через набор заранее определенных сообщений и строковых констант. Во многом подобно тому, как пользовательскими интерфейсными службами используются сообщения окон, интерфейс MCI обеспечивает набор командных сообщений и строк, которые можно применить для манипуляции MCI-устройствами. Многие из этих сообщений предлагают соответствующие командные строки, которые можно употребить для повышения читабельности. Однако здесь мы ограничимся рассмотрением командных сообщений MCI; константы сообщений определены в заголовочном файле `mmsystem.h`, доступ к которому можно получить с помощью C++Builder.

Подобно тому, как API-функция `SendMessage()` используется для отправки сообщений окнам, функция `mciSendCommand()` применяется для отправки командных сообщений MCI-устройствам.

```
MCIERROR mciSendCommand(
    MCIDEVICEID IDDevice,
    UINT uMsg,
```

```
    DWORD fdwCommand,
    DWORD dwParam
   );
```

При работе с функцией mciSendCommand() в качестве параметра IDDevice часто используется идентификатор устройства. Его назначение аналогично назначению параметра hWnd функции SendMessage(). А именно, функция должна "знать", к какому MCI-устройству нужно отправлять сообщение. Идентификатор устройства возвращается, когда устройство открывается посредством сообщения MCI_OPEN.

Декодирование констант ошибок

Функция mciSendCommand() возвращает 32-разрядное значение, по которому можно судить об успешном или неудачном выполнении операции. В случае успешного выполнения оно устанавливается равным значению MMSYSERR_NOERROR, которое в заголовочном файле mmsystem.h определяется равным нулю. В случае возникновения ошибки возвращаемое значение устанавливается равным одной из заданных констант ошибок. Поскольку эти значения малопонятны для пользователя (или разработчика), интерфейс MCI предоставляет функцию mciGetErrorString(), с помощью которой коды ошибок можно преобразовать в значащие сообщения (сравнимые с FormatMessage). Использование этой функции демонстрируется в листинге 25.15.

Листинг 25.15. Декодирование связанных с интерфейсом MCI ошибок с помощью функции mciGetErrorString()

```
bool mciCheck(DWORD AErrorNum, bool AReport = true)
{
    if (AErrorNum == MMSYSERR_NOERROR) return true;
    if (AReport)
    {
        char buffer[MAXERRORLENGTH];
        mciGetErrorString(AErrorNum, buffer, MAXERRORLENGTH);
        MessageBox(NULL, buffer, "MCI Error", MB_OK);
    }
    return false;
}
```

Работа с MCI-устройствами

Первый шаг в работе с MCI-устройствами — открыть или инициализировать устройство. Как упоминалось выше, эта задача выполняется с помощью сообщения MCI_OPEN. Поскольку нас интересует идентификатор устройства, мы посылаем это сообщение, используя для параметра IDDevice значение NULL. В случае успешного выполнения идентификатор открытого устройства возвращается в виде данного-члена wDeviceID соответствующей структуры MCI_OPEN_PARMS (структура MCI_OPEN_PARMS задается в качестве параметра dwParam).

```
typedef struct tagMCI_OPEN_PARMS {
    DWORD       dwCallback;
    MCIDEVICEID wDeviceID;
    LPCSTR      lpstrDeviceType;
    LPCSTR      lpstrElementName;
```

```
    LPCSTR      lpstrAlias;
}  MCI_OPEN_PARMS, *PMCI_OPEN_PARMS, *LPMCI_OPEN_PARMS;
```

Обычно данное-член `lpstrDeviceType` установлен равным `NULL`, а данному-члену `lpstrElementName` присваивается имя файла. Это наиболее устойчивый к ошибкам вариант, поскольку он позволяет интерфейсу MCI выполнять автоматический выбор типа, т.е. соответствующее устройство будет выбрано в соответствии с типом заданного файла. В случаях, когда используется явное задание данного-члена `lpstrDeviceType`, ему обычно присваивается строковое значение, соответствующее типу нужного устройства. Например, чтобы открыть устройство для проигрывания (чтения) аудиодисков, можно указать значение `cdaudio`. В число других строковых идентификаторов входят следующие: `avivideo`, `dat`, `digitalvideo`, `mmmovie`, `other`, `overlay`, `scanner`, `sequencer`, `vcr`, `videodisc` и `waveaudio`. Однако необходимо подчеркнуть, что если для устройства поддержка не предусмотрена, то лучше всего позволить интерфейсу MCI выполнить автоматический выбор типа. Это особенно важно в случае, когда используется новая технология, для которой еще нет встроенного идентификатора типа (например, для формата MP3). В листинге 25.16 демонстрируется использование сообщения `MCI_OPEN` посредством простой функции-контейнера.

Листинг 25.16. Использование командного сообщения `MCI_OPEN`

```
bool mciOpen(MCIDEVICEID& ADevID, const char* AFileName,
    const char* ADevType = NULL)
{
    MCI_OPEN_PARMS mop;
    memset(&mop, 0, sizeof(MCI_OPEN_PARMS));
    mop.lpstrElementName = const_cast<char*>(AFileName);
    mop.lpstrDeviceType = const_cast<char*>(ADevType);

    DWORD flags = 0;
    if (AFileName) flags = flags | MCI_OPEN_ELEMENT;
    if (ADevType) flags = flags | MCI_OPEN_TYPE;
    if (mciCheck(mciSendCommand(NULL, MCI_OPEN, flags,
                                reinterpret_cast<DWORD>(&mop))))
    {
        ADevID = mop.wDeviceID;
        return true;
    }
    return false;
}
```

После открытия MCI-устройства и считывания идентификатора нужно установить временной формат этого устройства. Этот аспект MCI очень привязан к типу устройства, поскольку определенные типы устройств могут поддерживать только определенные временные форматы. Например, задание номера дорожки действительно для устройств чтения компакт-дисков, но совершенно неприемлемо для устройств проигрывания музыкальных дисков (wave form audio devices).

Временной формат для устройства можно установить посредством командного сообщения `MCI_SET`. В общем случае лучше всего использовать формат `MCI_FORMAT_MILLISECONDS`, который поддерживается всеми устройствами. Пример функции-контейнера, которая использует сообщение `MCI_SET`, приведен в листинге 25.17.

Листинг 25.17. Использование командного сообщения MCI_SET

```
bool mciSetTimeFormat(MCIDEVICEID ADeviceID, DWORD ATimeFormat)
{
    MCI_SET_PARMS msp;
    memset(&msp, 0, sizeof(MCI_SET_PARMS));
    msp.dwTimeFormat = ATimeFormat;

    return mciCheck(mciSendCommand(ADeviceID, MCI_SET,
            MCI_SET_TIME_FORMAT, reinterpret_cast<DWORD>(&msp)));
}
```

После корректной установки формата с устройством можно работать практически так же, как с его физическим аналогом. Например, для воспроизведения звуковой информации используйте сообщение MCI_PLAY. Для перемотки или скоростного перехода используйте сообщение MCI_SEEK. Для приостановки устройства используйте сообщение MCI_PAUSE. Аналогично, сообщение MCI_STOP используется для останова устройства, а сообщение MCI_CLOSE закроет его. Чтобы получить полный список сообщений, обратитесь по адресу: http://msdn.microsoft.com/library/psdk/multimed/mci_7vvt.htm. Использование этих сообщений демонстрируют функции-контейнеры, представленные в листинге 25.18. Обратитесь также к прилагаемому к книге компакт-диску, чтобы увидеть еще более выразительный пример их использования (проект Proj_mp3Demo.bpr в папке MP3Demo, и в частности исходный файл MCIManip.cpp).

Листинг 25.18. Использование сообщений MCI_PLAY, MCI_SEEK, MCI_PAUSE, MCI_STOP и MCI_CLOSE

```
bool mciPlay(MCIDEVICEID ADeviceID, DWORD AStart, DWORD AStop)
{
    MCI_PLAY_PARMS mpp;
    memset(&mpp, 0, sizeof(MCI_PLAY_PARMS));
    mpp.dwFrom = AStart;
    mpp.dwTo = AStop;

    DWORD flags = 0;
    if (static_cast<int>(AStart) >= 0 && static_cast<int>(AStop) >= 0)
        flags = MCI_FROM | MCI_TO;

    return mciCheck(mciSendCommand(ADeviceID, MCI_PLAY | flags,
                    NULL, reinterpret_cast<DWORD>(&mpp)));
}

bool mciSeek(MCIDEVICEID ADeviceID, DWORD APos)
{
    MCI_SEEK_PARMS msp;
    memset(&msp, 0, sizeof(MCI_SEEK_PARMS));
    msp.dwTo = APos;

    return mciCheck(mciSendCommand(ADeviceID, MCI_SEEK, MCI_TO,
                    reinterpret_cast<DWORD>(&msp)));
}
```

```
bool mciPause(MCIDEVICEID ADeviceID)
{
    return mciCheck(mciSendCommand(ADeviceID, MCI_PAUSE, 0, 0));
}

bool mciStop(MCIDEVICEID ADeviceID)
{
    return mciCheck(mciSendCommand(ADeviceID, MCI_STOP, 0, 0));
}

void mciClose(MCIDEVICEID ADeviceID)
{
    mciCheck(mciSendCommand(ADeviceID, MCI_CLOSE, 0, NULL));
}
```

Чтение статуса устройства

Часто возникает необходимость обеспечить обратную связь с пользователем в отношении статуса устройства. Например, если вы собираетесь создать плейер компакт-дисков, то, вероятно, захотите информировать пользователя о текущей дорожке, ее длине и текущей позиции на дорожке. В общем случае вам придется указать режим работы устройства (воспроизведение, пауза, останов и т.д.), и тогда пользователь будет знать основные возможности данного устройства. Такая информация считывается с помощью сообщения `MCI_STATUS`, как показано в листинге 25.19.

Листинг 25.19. Использование сообщения `MCI_STATUS`

```
bool mciStatus(MCIDEVICEID ADeviceID, DWORD AQueryGroup,
             DWORD AQueryItem, DWORD AQueryTrack, DWORD& AResult)
{
    MCI_STATUS_PARMS msp;
    memset(&msp, 0, sizeof(MCI_STATUS_PARMS));
    msp.dwItem = AQueryItem;
    msp.dwTrack = AQueryTrack;

    if (mciCheck(mciSendCommand(ADeviceID, MCI_STATUS,
                    AQueryGroup, reinterpret_cast<DWORD>(&msp))))
    {
        AResult = msp.dwReturn;
        return true;
    }
    return false;
}
```

Проект `Proj_CDDemo.bpr` в папке `CDDemo` на прилагаемом к книге компакт-диске представляет собой пример проигрывателя компакт-дисков, который демонстрирует все методы, описанные в этих разделах.

Существует множество констант, которые можно использовать в качестве параметра `AQueryGroup`, причем каждая из них имеет соответствующий набор констант, которые могут быть присвоены параметрам `AQueryItem` и `AQueryTrack`. Например, чтобы прочитать

общее количество дорожек на музыкальном компакт-диске, укажите в качестве параметра AQueryGroup значение MCI_STATUS_ITEM, а в качестве параметра AQueryItem значение MCI_STATUS_NUMBER_OF_TRACKS. Для определения текущего номера дорожки установите параметр AQueryGroup равным значению MCI_STATUS_ITEM, а параметр AQueryItem — значению MCI_STATUS_CURRENT_TRACK. Обратите также внимание, что при считывании длины, позиции, дорожки или кадра информации формат возвращаемых данных зависит от текущего временного формата устройства. Для получения полного списка флагов статуса обратитесь по адресу: http://msdn.microsoft.com/library/psdk/multimed/mci_7vvt.htm.

MCI-уведомления

Коль у вас появилось средство считывания информации об устройстве, вам теперь нужно знать, когда выполнять такой запрос. К сожалению, интерфейс MCI предоставляет ограниченную схему уведомления, в которой предусмотрено только два уведомляющих сообщения: MM_MCINOTIFY и MM_MCISIGNAL. Последнее используется только для цифровых видеоустройств, а первое отправляется лишь после завершения операции. Например, если для того, чтобы приступить к воспроизведению звукового waveform-файла, вы используете функцию-контейнер mciPlay() (см. листинг 25.18), сообщение MM_MCINOTIFY будет послано только тогда, когда завершится воспроизведение файла, или этот режим будет отработан каким-то другим образом. В частности, это сообщение оправляется окну, дескриптор которого задается с помощью данного-члена dwCallback структуры, заданной в качестве параметра dwParam функции mciSendCommand(). В таком случае нужно модифицировать эту функцию-контейнер для приема параметра hWnd. Обработку сообщения MM_MCINOTIFY демонстрирует проект, который находится на прилагаемом к книге компакт-диске (Proj_mp3Demo.bpr в папке MP3Demo).

В большинстве случаев сообщения MM_MCINOTIFY оказывается вполне достаточно для определения статуса режима. Например, вы можете предоставить дескриптор для сообщения MM_MCINOTIFY, в котором будет обновляться разрешенное состояние кнопок воспроизведения, паузы или останова. Однако для считывания информации, связанной с таким часто обновляемым атрибутом, как текущая позиция, сообщение MM_MCINOTIFY не подходит. В этой ситуации необходимо опрашивать устройство через регулярные интервалы. Эта задача обычно выполняется в ответ на сообщения таймера. В одних случаях достаточно использовать сообщения системного таймера; в других — рекомендуется использовать мультимедийные таймерные службы. За дополнительной информацией обращайтесь по адресу: http://msdn.microsoft.com/library/psdk/multimed/mmtime_4msz.htm.

Заключительные замечания по поводу MCI

Итак, какие типы файлов поддерживает интерфейс MCI? Это зависит от аудио/видео кодеров-декодеров, которые инсталлированы на конечной платформе. Многие из этих кодеров-декодеров устанавливаются при установке более новой версии Microsoft Media Player. Основное проверенное практикой правило гласит, что если Media Player может поддерживать конкретный формат файла, то же может и MCI. И в самом деле, Media Player сам опирается в основном на интерфейс MCI.

В состав компакт-диска, прилагаемого к книге, включены два демонстрационных проекта, связанных с MCI: Proj_MP3Demo.bpr в папке MP3Demo и Proj_VideoDemo.bpr в папке VideoDemo. Первый представляет собой простой аудиоплейер MP3, который может также поддерживать RIFF-ориентированные музыкальные (waveform audio) файлы. Во втором демонстрируется использование интерфейса MCI для отображения видеофайлов (AVI, MPEG). Опять подчеркнем, что форматы файлов, реально поддерживаемые обоими демонстрационными проектами, ограничены установленными в данный момент кодерами-декодерами. Для

получения более подробной информации обратитесь к комментариям, приведенным в начале исходного кода этих проектов.

Несмотря на то что MCI — самый простой из всех мультимедиа-интерфейсов, его возможности весьма ограничены. Например, при воспроизведении файла через MCI вы никогда не получите доступ к потоку данных, связанному с файлом. Это особенно неприятно, если ваше приложение должно выполнять обработку сигналов или преобразование форматов. В этом случае вам придется отказаться от услуг MCI и работать с другими мультимедийными интерфейсами. Для расширения аудиовозможностей обычно используется интерфейс Waveform-Audio Interface.

Интерфейс Waveform Audio Interface

Мультимедийная служба Windows обеспечивает Waveform Audio Interface (waveform API), чтобы приложения могли управлять вводом и выводом звуковой информации (waveform audio). Этот интерфейс предоставляет приложению прямой доступ к звуковому буферу, поэтому в случае необходимости поддержки других форматов достаточно лишь выполнить простое преобразование. Например, многие коммерческие приложения, которые обеспечивают поддержку формата MP3, делают это именно через звуковой (waveform) API-интерфейс. Более того, в ситуациях, когда необходима обработка сигналов, прямой доступ к звуковому буферу является решающим фактором. Например, если вы заинтересованы в создании аудиоплейера с возможностями графической коррекции, вам нужно будет обрабатывать содержимое звукового буфера до отправки его на выходное устройство.

Вспомните, что данному-члену `lpstrElementName` структуры `MCI_OPEN_PARMS` обычно присваивается имя мультимедийного файла, подлежащего воспроизведению. В этом случае интерфейс MCI автоматически выполняет задачу открытия и загрузки файла. Однако звуковой интерфейс API не имеет такой структуры, что обязывает вас использовать другие средства ввода/вывода файлов. Например, одно потенциальное решение — открыть файл вручную с помощью класса VCL `TFileStream`. В этом случае для успешной работы нужно обладать достаточным опытом работы со спецификацией звукового формата файлов (RIFF). Другой подход состоит в использовании мультимедийных служб ввода/вывода файлов, которые совершенно правомочны во многих ситуациях. Однако, поскольку эти службы столь универсальны, работа со звуковыми (waveform audio) файлами оказывается почти такой же сложной, как и использование метода “вручную”. К счастью, в Windows предусмотрены службы AVIFile, набор функций и макросов, специально разработанных для работы со звуковыми (waveform audio) и AVI-файлами. А раз так, то давайте отойдем на время от звукового (waveform) интерфейса API и рассмотрим службы AVIFile.

Открытие и закрытие аудиофайлов

Аудиофайл (waveform) — это разновидность RIFF-файла (Resource Interchange File Format), аудиосодержимое которого определенным образом синхронизировано. Это, по сути, все, что вам нужно знать. Как упоминалось выше, вам необязательно вникать в специфику самого формата файла — достаточно использовать функции и макросы AVIFile. Эти функции и макросы, представленные в заголовочном файле `vfw.h` и библиотеке динамической компоновки `avifil32.dll`, обеспечивают удобные средства работы с аудиофайлами и потоками.

Прежде чем использовать службы AVIFile, необходимо инициализировать библиотеку `avifil32.dll` с помощью функции `AVIFileInit()`. Аналогично, по завершении работы библиотеку следует освободить с помощью функции `AVIFileExit()`.

Функции AVIFile для обработки файловых и потоковых операций опираются на технологию OLE, поэтому необходимо позаботиться о средствах контроля ошибок. Для функций, ко-

торые возвращают значение стандартного типа STDAPI, можно использовать макрос SUCCEEDED, как в следующей функции-контейнере.

```
bool wavCheck(HRESULT AErrorCode)
{
    return SUCCEEDED(AerrorCode);
}
```

Давайте начнем знакомство со службами AVIFile с выполнения простейших задач, а именно с открытия waveform-аудиофайла. Это реализуется посредством функции AVIFileOpen().

```
bool wavOpenFile(PAVIFILE& ApFile, const char* AFileName, unsigned int AMode)
{
    return wavCheck(AVIFileOpen(&ApFile, AFileName, AMode, NULL));
}
```

Параметр AMode задает режим доступа, и его значениями могут быть те же самые связанные с доступом константы, которые используются с функцией API OpenFile() (например, OF_READ). Параметр ApFile принимает указатель на структуру AVIFILE, которая просто хранит адрес интерфейса обработчика файлов. Поскольку этот интерфейс освобождается только при обнулении счетчика ссылок, важно декрементировать его, когда интерфейс больше не нужен. Это реализуется с помощью функции AVIFileClose().

```
void wavCloseFile(PAVIFILE& ApFile)
{
    AVIFileClose(ApFile);
    ApFile = NULL;
}
```

Работа с аудиопотоками

Несмотря на то что открытие и закрытие аудиофайлов довольно тривиально, но для получения какой-либо полезной информации вам придется иметь дело с интерфейсом обработчика потоков. Эта задача уже несколько сложнее. Вспомним, что тип PAVIFILE предназначен для хранения указателя на интерфейс обработчика файлов. Аналогично, указатель на интерфейс обработчика потоков хранится в переменной типа PAVISTREAM. Указатель на интерфейс обработчика потоков можно получить с помощью функции AVIFileGetStream(). Чтобы освободить этот интерфейс, воспользуйтесь функцией AVIStreamRelease(). Использование этих AVIFile-функций представлено в листинге 25.20.

Листинг 25.20. Использование функций AVIFileGetStream и AVIStreamRelease

```
bool wavOpenStream(PAVISTREAM& ApStream, PAVIFILE ApFile)
{
    return wavCheck(AVIFileGetStream(ApFile, &ApStream, streamtypeAUDIO, 0));
}

void wavCloseStream(PAVISTREAM& ApStream)
{
    AVIStreamRelease(ApStream);
    ApStream = NULL;
}
```

Работа с интерфейсом обработчика потоков подобна работе с классом TMemoryStream или одним из классов-потомков basic_streambuf. Однако в вашем распоряжении есть несколько функций, специально разработанных для работы с waveform- и AVI-файлами. Например, функцию AVIStreamInfo() можно использовать для считывания информации о содержимом потока. Эта функция заполняет структуру AVISTREAMINFO соответствующей информацией.

```
bool wavGetStreamInfo(PAVISTREAM ApStream, AVISTREAMINFO& AStreamInfo)
{
    return wavCheck(AVIStreamInfo(ApStream, &AStreamInfo,
                                  sizeof(AVISTREAMINFO)));
}
```

Работая с аудиопотоком, необходимо знать формат самих данных. Для аудиофайлов эта информация передается через данные-члены структуры WAVEFORMATEX. Эта структура просто используется для описания способа хранения звуковых выборок в соответствующих аудиоданных. В работе с аудиопотоками чрезвычайно полезной является функция AVIStreamReadFormat(). Назначение этой функции — заполнять данные-члены структуры WAVEFORMATEX на основе информации из потока. Макрос AVIStreamFormatSize() дополняет эту функцию, сообщая размер упомянутой структуры. Использование функции AVIStreamReadFormat() и макроса AVIStreamFormatSize() продемонстрировано в листинге 25.21.

Листинг 25.21. Использование функции AVIStreamReadFormat() и макроса AVIStreamFormatSize()

```
long wavCalcFormatStructSize(PAVISTREAM ApStream)
{
    long required_bytes = 0;
    AVIStreamFormatSize(ApStream, 0, &required_bytes);
    return required_bytes;
}

bool wavReadFormatStruct(PAVISTREAM ApStream, WAVEFORMATEX& ApFormatStruct)
{
    memset(&ApFormatStruct, 0, sizeof(WAVEFORMATEX));
    long size = wavCalcFormatStructSize(ApStream);
    return wavCheck(AVIStreamReadFormat(ApStream, 0, &ApFormatStruct,  &size)
        );
}
```

Подобно функциям-членам TMemoryStream::Read() и basic_ifstream::read(), служба AVIFile обеспечивает средства, с помощью которых приложение может прочитать содержимое потока. Важное место здесь отводится буферу аудиоданных, доступ к которому, как будет показано ниже, является решающим фактором для конечного результата при использовании звукового API-интерфейса. Функция AVIStreamRead() используется для чтения аудиоданных из потока в буфер, определенный приложением. Аналогично, функция AVIStreamWrite() используется для записи данных из буфера в поток. Использование этих функций продемонстрировано в листинге 25.22.

Листинг 25.22. Чтение и запись аудиоданных в поток и из него

```
long wavCalcBufferSize(PAVISTREAM ApStream)
{
    long required_bytes = 0;

    AVISTREAMINFO StreamInfo;
    if (wavGetStreamInfo(ApStream, StreamInfo))
    {
        required_bytes = StreamInfo.dwLength * StreamInfo.dwScale;
    }
    return required_bytes;
}

long wavReadStream(PAVISTREAM ApStream, long AStart,
                                  long ANumBytes, char* ABuffer)
{
    long bytes_read = 0;

    AVISTREAMINFO StreamInfo;
    if (wavGetStreamInfo(ApStream,  StreamInfo))
    {
        long num_samples = static_cast<float>(ANumBytes) /
                        static_cast<float>(StreamInfo.dwScale);
        AVIStreamRead(ApStream, AStart, num_samples, ABuffer,
                                  ANumBytes, &bytes_read, NULL);
    }
    return bytes_read;
}

long wavWriteStream(PAVISTREAM ApStream, long AStart,
                                  long ANumBytes, char* ABuffer)
{
    long bytes_written = 0;

    AVISTREAMINFO StreamInfo;
    if (wavGetStreamInfo(ApStream, StreamInfo))
    {
        long num_samples = static_cast<float>(ANumBytes) /
                        static_cast<float>(StreamInfo.dwScale);
        AVIStreamWrite(ApStream, AStart, num_samples, ABuffer,
                ANumBytes, AVIIF_KEYFRAME, NULL, &bytes_written);
    }
    return bytes_written;
}
```

Функцию-контейнер `wavCalculateBufferSize()` можно сравнить со свойством `TMemoryStream::Size`. Она использует функцию `wavGetStreamInfo()` для вычисления размера аудиобуфера. Обратите также внимание на то, что поскольку интерфейс обработчика потоков внутренне связан с интерфейсом обработчика файлов, при закрытии потока любые данные, ко-

торые записываются в поток, будут автоматически записаны в файл. Если же вас интересуют манипуляции лишь над содержимым потока, вам придется создать вторичный поток, который не связан с конкретным файлом. За дополнительной информацией по этой задаче обращайтесь по адресу: http://msdn.microsoft.com/library/psdk/multimed/avifile_4alv.htm.

Теперь, когда у вас в руках есть средства, с помощью которых вы можете манипулировать аудиофайлами, давайте вернемся к рассмотрению звукового API и исследуем средства, с помощью которых можно добиться звучания наших аудиоданных. Как и в случае интерфейса MCI, функции и структуры звукового API объявляются и определяются в заголовочном файле mmsystem.h.

Открытие и закрытие аудиоустройств

Звуковой интерфейс API содержит функцию waveOutOpen(), предназначенную для открытия аудиоустройств. Эта функция требует задания инициализированной структуры WAVEFORMATEX и при успешном выполнении присваивает переменной HWAVEOUT дескриптор открытого устройства. Аналогично, функция waveOutClose() используется для закрытия аудиоустройств. Использование этих двух функций показано в листинге 25.23.

Листинг 25.23. Открытие и закрытие аудиоустройств

```
bool wavPlayOpen(HWAVEOUT& AHWavOut, long ACallback,
                         DWORD ANotifyInstance, DWORD AOpenFlags,
                         WAVEFORMATEX& AFormatStruct)
{
    return wavCheck(
        waveOutOpen(&AHWavOut, WAVE_MAPPER, &AFormatStruct,
                          ACallback, ANotifyInstance, AOpenFlags)
        );
}

void wavPlayClose(HWAVEOUT AHWavOut)
{
    waveOutReset(AHWavOut);
    waveOutClose (AHWavOut);
}
```

Параметры ACallback, ANotifyInstance и AOpenFlags можно использовать для задания средств уведомления, но об этом чуть ниже. Для параметра AFormatStruct можно использовать функцию-контейнер wavReadFormatStruct() (см. листинг 25.21).

Открыв выходное устройство, используйте функцию waveOutWrite() для инициализации воспроизведения. В частности, передайте этой функции указатель на буфер аудиоданных, которые она будет оправлять открытому драйверу устройства вывода. Однако это устройство должно знать размер аудиоблока, подлежащего приему. Поэтому функция waveOutWrite() не может принимать простой буфер данных; ей нужен адрес структуры WAVEHDR.

Среди прочего структура WAVEHDR хранит указатель на буфер аудиоданных в своем данном-члене lpData, а также длину этого буфера в данном-члене dwBufferLength. Чтобы обеспечить совместимость с устройством вывода, необходимо позволить драйверу подготовить вашу структуру WAVEHDR до ее передачи функции waveOutWrite(). Эта задача выполняется с помощью функции waveOutPrepareHeader(). Аналогично, после того, как драйвер устройства завершит воспроизведение аудиоблока, необходимо освободить заголовок с помощью

функции wavOutUnprepareHeader(), причем это нужно сделать до освобождения выделенной ранее памяти. Использование этих функций показано в листинге 25.24.

Листинг 25.24. Инициализируется и завершение воспроизведения аудиоблока

```
bool wavPlayBegin(HWAVEOUT AHWavOut, WAVEHDR& AWavHdr)
{
    if (wavCheck(waveOutPrepareHeader(AHWavOut, &AWavHdr, sizeof(WAVEHDR))))
    {
        return wavCheck(
            waveOutWrite(AHWavOut, &AWavHdr, sizeof(WAVEHDR))
            );
    }
    return false;
}

void wavPlayEnd(HWAVEOUT AHWavOut, WAVEHDR& AWavHdr)
{
    waveOutReset(AHWavOut);
    waveOutUnprepareHeader(AHWavOut, &AWavHdr, sizeof(WAVEHDR));
}
```

Давайте закрепим рассмотренное с помощью простого примера, который демонстрирует один из способов воспроизведения аудиофайла. Вспомним, что, поскольку звуковой API не предоставляет никаких встроенных средств загрузки аудиоданных с диска, нам сначала нужно воспользоваться функциями-контейнерами AVIFile. Прочитав блок аудиоданных из потока в буфер, можно использовать функции звукового API для управления процессом воспроизведения. Код этого примера представлен в листинге 25.25. Сам проект, Proj_DSPDemo.bpr, можно найти на прилагаемом к книге компакт-диске (в папке DSPDemo).

Листинг 25.25. Воспроизведение дискретного аудиофайла

```
const long MAX_BLOCK_SIZE = 6000 * 1024;

if (!OpenDialog1->Execute()) return;
const char* filename = OpenDialog1->FileName.c_str();

PAVIFILE pFile = NULL;
if (wavOpenFile(pFile, filename, OF_READ))
{
    PAVISTREAM pStream = NULL;
    if (wavOpenStream(pStream, pFile))
    {
        long block_size = wavCalcBufferSize(pStream);
        if (block_size < MAX_BLOCK_SIZE)
        {
            char* buffer = new char[block_size];
            if (wavReadStream(pStream, 0, block_size, buffer)
                == block_size)
            {
                QuantizeBuffer(buffer, block_size);
```

```
                WAVEFORMATEX FormatStruct;
                if (wavReadFormatStruct(pStream, FormatStruct))
                {
                    HWAVEOUT HWavOut;
                    if (wavPlayOpen(HWavOut, NULL, NULL, NULL, FormatStruct))
                    {
                        WAVEHDR WavHdr;
                        memset(&WavHdr, 0, sizeof(WAVEHDR));
                        WavHdr.lpData = buffer;
                        WavHdr.dwBufferLength = block_size;

                        if (wavPlayBegin(HWavOut, WavHdr))
                        {
                            ShowMessage("Playing: " + AnsiString(filename));
                            wavPlayEnd(HWavOut, WavHdr);
                        }
                        wavPlayClose(HWavOut);
                    }
                }
            }
            delete [] buffer;
        }
        wavCloseStream(pStream);
    }
    wavCloseFile(pFile);
}

void QuantizeBuffer(char* ABuffer, long ABufferLength)
{
    short int min_val = 0, max_val = 0;
    for (int index = 0; index < ABufferLength; ++index)
    {
        if (ABuffer[index] < min_val) min_val = ABuffer[index];
        if (ABuffer[index] > max_val) max_val = ABuffer[index];
    }

    for (int index = 0; index < ABufferLength; ++index)
    {
        if (ABuffer[index] < 0) ABuffer[index] = min_val;
        if (ABuffer[index] > 0) ABuffer[index] = max_val;
    }
}
```

Обратите внимание (см. листинг 25.25), что мы ограничили размер блока аудиоданных. И в самом деле, чтобы не истощать системные ресурсы, не стоит загружать слишком большую порцию файла. При работе с большими по размеру блоками аудиоданных из потока читаются небольшие порции данных, которые затем последовательно посылаются драйверу. Это значит, что после того, как драйвер справится с текущим буфером, мы тут же снабдим его новой информацией. Однако чтобы гарантировать успешную работу такого метода, нам понадобятся средства, которые позволят определить момент завершения обработки буфера. Поскольку функция `waveOutWrite()` немедленно возвращает управление, мы не в состоянии узнать, ко-

гда драйвер завершит воспроизведение. Вот тут-то нам и пригодится параметр ACallback функции wavPlayOpen(). В частности, мы можем присвоить этому параметру дескриптор окна или события, идентификатор потока или даже адрес конкретной функции обратного вызова. И тем самым установим ключевые средства уведомления. Для получения информации по сообщению MM_MON_DONE обращайтесь по адресу: http://msdn.microsoft.com/library/psdk/multimed/mmmsg_6g2t.htm.

Заключительные замечания по интерфейсу Waveform Audio Interface

Мы рассмотрели множество мультимедийных функций, структур, сообщений и макросов, но нерассмотренных — еще больше. Например, мультимедийная система Windows также предоставляет специальный интерфейс для управления MIDI- и AVI-воспроизведением. Более того, каждый из этих интерфейсов, включая интерфейс Waveform Audio Interface и MCI, обеспечивает услуги по вводу и выводу звуковых сигналов. Существует даже интерфейс для работы со звуковыми микшерами. Для получения более полной информации обращайтесь по адресу: http://msdn.microsoft.com/library/psdk/multimed/mixer_10xf.htm.

Резюме

В этой главе мы рассмотрели несколько методов, с помощью которых ваше приложение может обеспечить мультимедийную поддержку. Мы рассмотрели основы графического интерфейса Windows с устройствами (GDI) и множество аспектов этого интерфейса, реализованных библиотекой VCL. Мы затронули использование классов TCanvas, TBrush, TPen и TFont в воспроизведении графических изображений. Мы также рассмотрели несколько форматов файлов изображений и использование классов TBitmap и TJPEGImage.

Затем мы остановились на теме обработки изображений, которая сама по себе довольно обширна. Мы рассмотрели ряд основных методов, существующих в этой области, и способов их реализации в среде C++Builder. Освоив этот материал, вы сможете пойти дальше и разработать собственные приложения обработки изображений.

Наконец, мы уделили внимание поддержке аудио- и видеофайлов путем использования интерфейса MCI. Были рассмотрены различные MCI-сообщения и их использование в управлении мультимедийными устройствами, а также использование служб AVIFile и интерфейса Waveform Audio Interface для управления и эффективного воспроизведения аудиофайлов.

Глава

26

Покорение новых вершин компьютерной графики с помощью интерфейсов DirectX и OpenGL

Вильям Вудбури

Одной из проблем программистов (особенно специализирующихся в области компьютерных игр) при переходе от DOS к графическим операционным системам (например, Microsoft Windows) стала производительность приложений реального времени, в которой, как им казалось, ощущался существенный проигрыш. Это проявилось в результате затрат, связанных с мультизадачностью и интерактивной графикой высокого разрешения, используемой в интерфейсе пользователя. Эти потери в производительности были компенсированы благодаря введению таких высокоскоростных мультимедийных API-интерфейсов, как OpenGL и DirectX.

OpenGL — это интерфейс высокого уровня с графической аппаратурой, обеспечивающий программную эмуляцию всех его функций (для компьютеров без специального оборудования). DirectX — это набор API-интерфейсов, который позволяет программистам получать доступ к аппаратуре низкого уровня в компьютере, работающем под управлением операционной системы Microsoft Windows. Эта глава посвящена использованию (в среде Borland C++Builder 5) интерфейса OpenGL, а также API-интерфейсов DirectDraw и DirectSound, входящих в состав DirectX.

Высокопроизводительная графика — неотъемлемая часть многих современных программных продуктов. Ее освоение не столь трудно, как кажется на первый взгляд. В этой главе раскрываются все основные требования, предъявляемые к отображению качественных трехмерных изображений и созданию анимации.

Введение в OpenGL

В этом разделе вы получите общее представление о том, что такое интерфейс OpenGL и для чего он нужен. OpenGL — это API для трехмерной графики, а проще говоря, набор библиотечных функций, упрощающих создание сложных графических изображений в режиме реального времени.

Сам по себе интерфейс OpenGL лишь определяет спецификацию для работы с функциями. Реальная их реализация может быть разной. Основными производителями являются компания Silicon Graphics, Inc. (SGI) и Microsoft (MS). Настоящая реализация была создана и поддерживается фирмой OpenGL Architecture Review Board (ARB), независимой организацией, созданной специально для этой цели.

OpenGL в целом не зависит от платформы. Если программист предпочитает использовать функции, утвержденные организацией ARB, его программу можно переносить на любую платформу, имеющую OpenGL-реализацию.

Реализации OpenGL будут использовать специальное графическое оборудование, если таковое присутствует. Это чрезвычайно полезно, поскольку большинство современных домашних и офисных компьютеров оборудовано трехмерными акселераторами. Но даже если такого оборудования нет, программные реализации, отличаясь высокой оптимизацией, работают очень быстро.

OpenGL становится мировым стандартом в графике реального времени. В каждом приличном пакете средств трехмерного моделирования обязательно используется OpenGL для отображения в режиме реального времени. Большинство новых компьютерных игр также опирается на этот стандарт.

OpenGL-программа приятна с эстетической точки зрения. В именах функций и переменных используются стандарты присваивания имен, которые легко понять и соблюдать.

OpenGL в сравнении с Direct3D

Direct3D можно расценивать как ответ Microsoft компании OpenGL. Это — трехмерный графический API, который может взаимодействовать с аппаратурой. Если вы собираетесь использовать Direct3D, имейте в виду, что он не обладает переносимостью, и существует лишь одна его реализация — в исполнении компании Microsoft.

На заметку

Не смешивайте работу с OpenGL с DirectDraw или Direct3D. Дело в том, что некоторые драйверы OpenGL используют эти API для имитации вызовов OpenGL на компьютерах, не оснащенных видеокартами OpenGL. Если DirectDraw или Direct3D используется в одной и той же программе с OpenGL, они могут вступить в противоречие и привести к непредсказуемым результатам.

Структура команд OpenGL

В OpenGL для создания примитива (объекта) в трехмерной сцене соблюдается общая структура команд.

Большинство команд OpenGL совершенно независимо от конкретных примитивов. Это значит, что их можно выдать для любого примитива OpenGL, и при этом будет достигнут аналогичный эффект. Примером этого факта может служить команда `glColor3f()`: в результате ее применения к примитиву OpenGL будет установлен заданный цвет.

Тип создаваемого примитива определяется параметрами функции `glBegin()`. Все, что входит в понятие реального рисования (спецификации точек) в OpenGL, происходит между операторами `glBegin()` и `glEnd()`.

Циклы рисования в среде C++Builder, реализуемые с помощью функции `OnIdle()`

Рисование сцены обычно происходит в цикле обновления. Это значит, что постоянно выполняется некоторый цикл, в котором с помощью интерфейса OpenGL рисуется определенная сцена, а затем все ее динамические части обновляются (например, перемещаются определенные объекты). Таким образом создаются анимационные эффекты.

Добиться этого программист может двумя следующими способами.

- Используя поток. Это довольно хороший метод, но многопотоковость сопровождается большими затратами.
- Используя функцию `OnIdle()`. Это более предпочтительный метод. `OnIdle()` — это виртуальная функция, которая вызывается, когда система находится в состоянии ожидания, и в это время никакие команды или сообщения не обрабатываются и не выдаются.

Функцию, на которую будет указывать функция `OnIdle()`, следует определить следующим образом, заменив имя *`IdleFunction()`* соответствующим именем функции.

```
void __fastcall TForm1::IdleFunction(TObject* Sender, bool &done)
{
```

Первой строкой в этой функции должна быть

```
done = false;
```

Это потребует от системы большего времени ожидания. Остальная часть функции определяется исключительно программистом. Обычно она включает код рисования, сопровождаемый переключением страниц, если используется двойная буферизация.

Функция `OnIdle()` устанавливается следующим образом.

```
Application->OnIdle = IdleFunction;
```

Эту установку можно выполнить в любое время для любой корректно определенной функции. Рекомендуется не устанавливать `OnIdle()` до тех пор, пока не инициализирован интерфейс OpenGL. Если обработчик содержит OpenGL-код, то он не будет выполняться корректно, а результаты могут быть совершенно непредсказуемыми.

Использование OpenGL

Большинство OpenGL-программ имеют общую структуру, которую можно разделить на несколько этапов.

1. Установка и инициализация. Этап создания контекстов рендеринга и устройств (это называется инициализацией OpenGL).
2. Установка среды рендеринга. Этап установки таких условий в виртуальной среде, как освещение, туман и модель обработки оттенков.
3. Преобразование примитивов. Этап определения позиции, ориентации и масштаба рисуемых точек. См. раздел "Этап 3: трехмерные преобразования" ниже в этой главе.
4. Рисование примитивов. Этап определения реальных точек и примитивов, которых они представляют. Здесь также готовится спецификация материалов и выполняется текстурирование.
5. Смена поверхности. Это касается только рисования с двойной буферизацией.

Этапы 2–5 обычно относятся к циклу рисования, как описано в разделе "Циклы рисования в среде C++Builder, реализуемые с помощью функции `OnIdle()`" выше в этой главе.

Этап 1: инициализация OpenGL

Прежде чем приступить к рисованию, необходимо выполнить инициализацию, чтобы подготовить операционную систему для работы с интерфейсом OpenGL. Для того чтобы OpenGL мог функционировать, нужно создать то, что называется *контекстом рендеринга*. Если это словосочетание звучит непривычно, попробуйте представить себе OpenGL в виде контекста некоторого устройства, который используется для хранения определенной информации, например, описывающей его текущее состояние.

На заметку

Windows-совместимые реализации OpenGL имеют дополнительный набор команд, предназначенных специально для взаимодействия с операционной системой. Эти команды называются *Wiggle-функциями*, поскольку имена всех этих функций начинаются с префикса *wgl*, который произошел от Win32 GL.

OpenGL может создать контекст рендерингана основе контекста устройства с помощью функции `wglCreateContext()`. Создание контекста рендеринга OpenGL показано в листинге 26.27.

Листинг 26.1. Инициализация OpenGL: функция `OpenGLInit()`

```
// Глобальные идентификаторы (объявляются либо в классе TForm1, либо глобально).
HDC hDC;
HGLRC hRC;

bool TForm1::OpenGLInit( void )
{
// Получаем контекст устройства из главного окна (TForm1).
    hDC = GetDC( Handle );

    // При возникновении ошибки функция GetDC возвратит NULL.
    if( hDC == NULL )
        return( false );
```

```
    // Об этой функции речь пойдет ниже.
    SetGLPixelFormat( hDC );

    // Здесь используем контекст устройства для создания контекста рендеринга.
    hRC = wglCreateContext( hDC );

    // При возникновении ошибки функция wglCreateContext возвратит значение NULL.
    if( hRC == NULL )
        return( false );
```

После того как контексты устройства и рендеринга созданы, они должны быть установлены таким образом, чтобы их могли использовать OpenGL и Windows. "Подмога" приходит со стороны еще одной *wgl*-функции, а именно функции `wglMakeCurrent()`.

```
    wglMakeCurrent( hDC, hRC );
    return( true );
}
```

Вероятно, вы обратили внимание на функцию `SetGLPixelFormat()` из предыдущего фрагмента программы. У нее очень важная роль: именно она устанавливает нужные для OpenGL разряды контекста устройства, чтобы интерфейс OpenGL мог должным образом контактировать с Windows.

Для этого требуется установить структуру `PIXELFORMATDESCRIPTOR`. Работа функции `SetGLPixelFormat()` показана в листинге 26.2.

Листинг 26.2. Установка пиксельного формата OpenGL: функция `SetGLPixelFormat()`

```
void SetGLPixelFormat( HDC hdc )
{
    int PixelFormatIndex;
    PIXELFORMATDESCRIPTOR PixelFormat=
    {
        sizeof( PIXELFORMATDESCRIPTOR ),        // Размер структуры.
                1,                              // Версия структуры.
        PFD_DRAW_TO_WINDOW |                    // Рисуем прямо в окно, а не
                                                // в битовый массив экрана.
        PFD_SUPPORT_OPENGL |                    // Позволяем контексту устройства
                                                // поддерживать вызовы OpenGL
                                                // (может пригодиться!).
        PFD_DOUBLEBUFFER,                       // Используем двойной буфер.
        PFD_TYPE_RGBA,                          // Используем цветовой режим RGBA.
        24,                                     // Используем 24-разрядный цвет.
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,  // Не используются
        32,                                     // Используем 32-разрядный z-буфер.
        0, 0,                                   // Не используются.
        PFD_MAIN_PLANE,                         // Рисуем на главную поверхность.
        0, 0, 0, 0                              // Не используются.
    } ;

    // Выбираем индекс соответствующего пиксельного формата.
    PixelFormatIndex=ChoosePixelFormat( hdc, &PixelFormat );
```

```
    // Устанавливаем пиксельный формат.
    SetPixelFormat( hdc, PixelFormatIndex, &PixelFormat );
}
```

Как работает рендеринг в OpenGL

Отображение трехмерных графических изображений может включать сложную группу процессов. Назначение OpenGL API — насколько возможно упростить эту сложность за счет инкапсуляции трудных понятий в виде простых в применении функций.

Окно просмотра

Для определения окна, в котором отображаются результаты работы OpenGL, программист использует окно просмотра, или окно проекции (viewport). Это просто двумерная область экрана, в котором OpenGL создает виртуальный трехмерный мир.

В изображении, которое OpenGL проецирует в окно проекции, всегда (в качестве некоторой отправной точки) используется начало координат трехмерного пространства (т.е. точка с координатами 0,0,0) с учетом направления вдоль оси Z.

На заметку Объяснение терминов и теоретические основы трехмерной графики можно найти в любом учебнике по соответствующий теме.

Используя OpenGL для отображения трехмерных изображений, программист сначала приводит систему локальных координат в преобразованные внешние координаты, а затем — в относительные. Под *преобразованными внешними координатами* понимаются точки в пространстве, к которым были применены матрицы преобразований, в результате чего их координаты (относительно объектного пространства, в котором они были созданы) были приведены к координатам относительно мирового пространства. Подробнее см. в разделе “Этап 3: трехмерные преобразования” ниже в этой главе или в учебнике по трехмерной графике.

Система локальных координат включает исходные непреобразованные точки в трехмерном пространстве. К этим точкам затем применяются различные преобразования (которые будут описаны ниже), чтобы перевести их в мировую, или глобальную (внешнюю), систему координат. Локальные координатные системы групп точек (объектов) необходимо затем корректно расположить относительно друг друга.

Эти координаты должны быть преобразованы таким образом, чтобы они выглядели корректными относительно позиции и ориентации зрителя. Это делается путем реверсирования преобразований, используемых для размещения зрителя относительно каждого объекта в сцене. Это эффективно перемещает зрителя в начало координат, сохраняя расположение всех других объектов по отношению к нему. Следовательно, когда *сцена* (собирательный термин для всех трехмерных объектов и артефактов в виртуальном пространстве) изображается в окне проекции, она отображается определенным образом по отношению к зрителю. На рис. 26.1 показана ориентация зрителя относительно глобальной системы координат.

Возможно, новичкам в трехмерной графике этот материал покажется непонятным и трудным для освоения. Не волнуйтесь, надеемся, все прояснится, когда доберемся до примеров. Визуализация всегда делает создание трехмерных графических изображений проще и понятнее. Под визуализацией иногда понимают способность представлять влияние преобразований на точки в трехмерном пространстве. Эту способность можно развить, поэтому тем, кто считает визуализацию чем-то недоступным для себя, паниковать не стоит. Если ваша программа будет написана корректно, вы увидите ее результаты на экране.

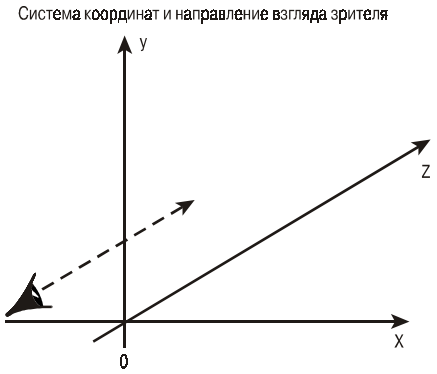


Рис. 26.1. Ориентация зрителя относительно глобальной системы координат

Установка окна проекции

Существуют две функции, которые программист может использовать для определения окна проекции. Одна из них устанавливает реальное окно проекции (окно в виртуальном мире), а другая определяет, как трехмерные данные отображаются в этом окне. Различают два основных метода отображения: *перспективный* и *ортографический*. В перспективном отображении все трехмерные координаты умножаются на матрицу, которая создает впечатление перспективы (объекты становятся меньше при более удаленном расстоянии от камеры). При ортографическом отображении точки наносятся на экран линейным способом, безотносительно к расстоянию. В этом случае создаются плоско-пространственные изображения, которые выглядят менее реалистично. Однако второй метод изображения необходим для таких технических программ, как пакеты CAD, в которых данные должны изображаться как есть.

В листинге 26.3 показана простая функция, которая может установить либо перспективное, либо ортографическое окно просмотра.

Листинг 26.3. Установка окна проекции: функция `SetViewport()`

```
void TForm1::SetViewport( bool Perspective )
{
    float w, h, Aspect;

    w=Width;
    h=Height;

    // Мы хотим избежать попыток деления на нуль.
    if(h==0)
        h=1;

    // Устанавливаем реальную область просмотра для всей клиентской области.
    glViewport(0, 0, w, h);

    // Устанавливаем матричный режим для проекции
    // (т.е. мы хотим, чтобы операции влияли только на матрицу проекции).
    glMatrixMode(GL_PROJECTION);
```

```
    // Загружаем единичную матрицу (эта матрица эквивалентна нулю).
    glLoadIdentity();

       // Если пользователь пытается создать окно перспективной проекции, то...
    if(Perspective)
    {
        // ...выполняем эту установку
        Aspect=(GLfloat)w/(GLfloat)h;
        gluPerspective(60.0f, Aspect, 1.0f, 1000.0f);
    }
    // в противном случае пользователь пытается создать
    // окно ортографической проекции, поэтому...
    else
    {
        // ...выполняем эту установку
        if(w<=h)
            glOrtho(-250.0f, 250.0f, -250.0f*h/w, 250.0f*h/w,
                                                  1.0f, 1000.0f);
        else
            glOrtho(-250.0f*w/h, 250.0f*w/h, -250.0f, 250.0f,
                                                  1.0f, 1000.0f);
    }

    // Используем матрицу преобразования (см. раздел, посвященный преобразованиям)

    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
}
```

В комментариях листинга присутствуют ссылки на матрицу. Матрица — это математический прием, который можно использовать для выполнения операций над векторами или другими матрицами. Эти операции в контексте трехмерной графики могут быть проекцией (как в листинге 26.3, матрица проекции) или преобразованием. Преобразование — это операция над трехмерными координатами, например перемещение (сдвиг или перенос), поворот или масштабирование. Преобразование рассматривается ниже в этой главе.

Функция `glViewport()` устанавливает реальное окно просмотра в клиентских координатах (в окне). Это своего рода канал, через который видна трехмерная сцена. Эта функция имеет следующий список параметров.

```
glViewport( int x, int y, int x2, int y2 )
```

Здесь `x`, `y` — координаты нижнего левого угла окна, а `x2`, `y2` координаты верхнего правого его угла.

Функция `glOrtho()` требует, чтобы была использована ортографическая матрица, и устанавливает способ влияния этой матрицы на данные.

```
glOrtho( int x, int y, int x2, int y2, float near, float far )
```

Здесь `x`, `y` — координаты нижнего левого угла объема просмотра, а `x2`, `y2` — координаты верхнего правого его угла. Данные внутри этого объема просмотра сплющиваются в определенное окно просмотра. Параметр `near` —самое короткое расстояние до зрителя, при котором трехмерные данные видимы, а `far` — самое длинное.

Функция gluPerspective() требует, чтобы была использована матрица перспективной проекции, и устанавливает способ влияния этой матрицы на данные. Эта функция имеет следующий список параметров.

```
gluPerspective( float fov, float aspect, float near, float far )
```

fov — поле просмотра, в котором данные видимы. Оно представляет собой "смотровой конус" видимости. Aspect — коэффициент сжатия, или отношение ширины к высоте. Near и far — параметры, аналогичные соответствующим параметром функции glOrtho().

Префикс glu в имени функции gluPerspective() означает, что эта функция принадлежит к пакету функций OpenGL.

Механизм управления режимами OpenGL

Пользователь управляет характером работы OpenGL путем изменения его *режимов*. Режим, или флаг режима, с точки зрения OpenGL — это внутренняя переменная, которая разрешает пользователю управлять различными аспектами процесса рендеринга. Большинство этих переменных — булевого типа (т.е. имеет либо истинное, либо ложное значение), но некоторые — вещественного (с плавающей запятой) или векторного типа (например, расположение источника света и цвет).

Установка и сброс режимов

Основными функциями, предназначенными для управления режимами булевого типа, являются glEnable() и glDisable(). Чтобы разрешить двумерное преобразование текстур, используйте следующий вызов функции.

```
glEnable( GL_TEXTURE_2D );
```

Для его запрещения достаточно сделать такой вызов.

```
glDisable( GL_TEXTURE_2D );
```

Получение информации о режиме

Чтобы узнать, установлен ли режим булевого типа, можно использовать функцию glIsEnabled().

```
if( glIsEnabled( GL_TEXTURE_2D ) ) ...
```

Этап 2: установка среды рендеринга (освещение и затенение)

Не будь света и тени, все в сцене выглядело бы плоским и неестественным. OpenGL позволяет легко установить освещение сцены с помощью нескольких простых команд. В следующем разделе описано, как освещение влияет на сцену, и рассмотрена модель, которая используется для имитации освещения и затенения.

Модели освещения и затенения

Чтобы отражать в реальном времени изменения в условиях виртуального освещения, необходимо динамически обновлять состояние затенения сцены. Это очень важно в создании реалистичной среды. Динамическое затенение достигается в OpenGL за счет использования виртуальных источников света, которые можно настроить на создание различных эффектов освещения.

Затенение многоугольника (плоская поверхность, составленная из трех или более точек в трехмерном пространстве) зависит от трех различных типов освещения: внешнего (общего), диффузного (рассеянного) и отраженного.

Внешний свет создает такой уровень освещения, который обычно присутствует в любой среде. Он состоит из многократно отраженного света, поэтому его источник не определен.

Диффузный и отраженный свет образуется в результате отражения от поверхности многоугольника света, приходящего от его источника. Диффузный — отражается в произвольных направлениях; поэтому он виден из любой точки, откуда видна освещенная сторона многоугольника. Отраженным считается свет, угол отражения (сфера действия) которого равен углу, под которым свет падает на многоугольник (рис. 26.2).

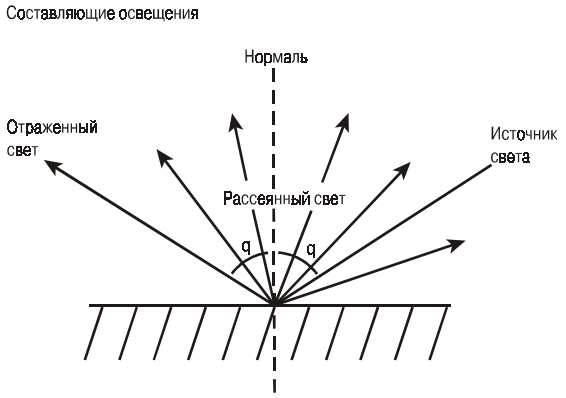


Рис. 26.2. Взаимодействие света с плоской поверхностью

Модели затенения OpenGL

Интерфейс OpenGL использует две модели затенения, часто называемые *плоскостной* (`flat model`) и *сглаживания* (`smooth model`). Обе модели оптимизируются для получения максимальной скорости и поэтому не применяют уравнение освещения к каждому отображаемому пикселю.

В модели сглаживания такое уравнение применяется к точкам, составляющим многоугольник (к трем точкам для треугольника, к четырем — для четырехугольника). Эти значения затем интерполируются для получения значений освещенности в каждой точке многоугольника. Этот метод хорошо себя зарекомендовал практически во всех случаях.

В плоскостной модели уравнение применяется один раз для многоугольника в целом, поэтому он весь оттеняется одним и тем же значением цвета. Этот метод приемлем, если многоугольник достаточно мал.

Модель сглаживания работает несколько медленнее, чем плоскостная, но ее результаты гораздо лучше, особенно в случаях, когда модель содержит несколько многоугольников (плоскостная модель тогда вообще вряд ли подойдет: все будет выглядеть очень искусственно).

Установка источников света для динамического затенения

Источники света устанавливаются с помощью набора функций `glLight()`.

Большинство функций интерфейса OpenGL названы с использованием следующего соглашения:

`gl<`*`FunctionName`*`>[n][t](...)`

Здесь *<FunctionName>* представляет собой базовое описание того, к чему эта функция применяется (например, `Vertex`, `Light`, `Normal`).

`[n]` — количество параметров, принимаемых функцией.

`[t]` — тип принимаемого параметра (например, `f` для типа `float`, `d` для типа `double`, `i` для типа `integer`). Если за `[t]` следует `v`, то функция принимает не отдельные аргументы, а `[n]` параметров в массиве. Приведем примеры OpenGL-функций, соблюдающих это соглашение: `glLightfv(...)`, `glVertex3f(...)` и `glNormal4fv(...)`.

Реальные параметры освещения устанавливаются в листинге 26.4.

Листинг 26.4. Установка параметров освещения

```
// Устанавливаем компонент внешнего освещения равным темно-серому.
float AmbientColor[4]={ 0.2f, 0.2f, 0.2f, 1.0f} ;
glLightfv(GL_LIGHT0, GL_AMBIENT, AmbientColor);

// Устанавливаем компонент диффузного освещения равным светло-серому.
float DiffuseColor[4]={ 0.8f, 0.8f, 0.8f, 1.0f} ;
glLightfv(GL_LIGHT0, GL_DIFFUSE, DiffuseColor);

// Устанавливаем компонент отраженного освещения равным ярко-белому.
// float SpecularColor[4]={ 1.0f, 1.0f, 1.0f, 1.0f} ;
glLightfv(GL_LIGHT0, GL_SPECULAR, SpecularColor);
Next, set the position of the light in your scene:
// Устанавливаем позицию источника света в начале координат.
float LightPosition[4]={ 0.0f, 0.0f, 0.0f, 0.0f} ;
glLightfv(GL_LIGHT0, GL_POSITION, LightPosition);
```

Обратите внимание, что параметры для цветовых составляющих источника света передаются в виде четырехэлементного массива чисел с плавающей запятой. Эти элементы соответствуют красной, зеленой, синей и альфа составляющим. Альфа-составляющая не используется при установке освещения, поэтому она просто устанавливается равной `1.0f`. Эти значения лежат в диапазоне между `0.0f` (отсутствие яркости) и `1.0f` (полная яркость). Если пользователь задаст значение, не попадающее в этот диапазон, оно будет заменено значением `0.0f` (при заданном значении ниже нижней границы) или значением `1.0f` (при заданном значении выше верхней границы). Это позволяет интерфейсу OpenGL оставаться независимым от устройств, поскольку он внутренне отрабатывает реальные цветовые значения, зависящие от контекстов устройства и рендеринга.

Элементами в массиве `LightPosition` являются координаты `x`, `y`, `z` и `w`. Координату `w` можно установить равной `0.0f` (ее описание выходит за рамки этой главы).

Еще несколько слов об освещении. Существуют различные типы света, но здесь мы описали метод создания освещения стандартного типа: ненаправленного света. В OpenGL можно также создать точечные источники света путем использования конусов и направленности освещения. (См. раздел “Дополнительные источники информации по OpenGL” ниже в этой главе.).

Этап 3: трехмерные преобразования

Можно, конечно, удовлетвориться и статической сценой, но без движения и анимации (а, значит, и трехмерной графики) вам не добиться грандиозного эффекта. Преобразование, говоря языком геометрии, — это любая операция, выполненная над точками, в результате которой меняется взаимное расположение этих точек. В разряд таких преобразований входят: масштабирование, поворот и перенос (сдвиг).

Конвейер преобразования (из трехмерных координат в пиксели)

В OpenGL (как и в большинстве, если не во всех трехмерных API) данные проходят три стадии. Все начинается с точек в локальной координатной системе. Локальная координатная система описывает исходные точки относительно фиксированного начала координат в сплошном объекте или контуре. Следовательно, каждый контур имеет собственную локальную систему координат. Эти точки затем преобразуются во внешнюю систему координат с помощью операций над матрицами. Внешняя система координат описывает каждую точку “с прицелом” ее расположения в сцене (виртуальном мире) относительно некоторого фиксированного начала координат. Преобразованные точки не отображаются до тех пор, пока не будут спроецированы в два измерения (поскольку компьютерные экраны плоские). OpenGL выполняет все эти преобразования автоматически при определенных окне просмотра и матрицах проекций. Последней стадией является отображение, которое также выполняется автоматически интерфейсом OpenGL с использованием заданных контекстов устройства и рендеринга.

Следовательно, программисту достаточно побеспокоиться об установке окна просмотра (см. раздел “Как работает рендеринг в OpenGL” выше в этой главе), подготовке точек (это делается с помощью набора функций `glVertex()`) и их преобразовании. В этом разделе мы остановимся на последнем шаге.

Три преобразования

К точкам в OpenGL применяются три основных преобразования: перенос (сдвиг), масштабирование и поворот. Сами преобразования применяются к точкам внутренне, от вас требуется лишь задать нужные точки, причем всегда в локальных координатах.

Перенос

На рис. 26.3 показан результат переноса.

Под переносом понимают перемещение точек, заданное определенным образом. Эта операция очень легко применяется с использованием метода `glTranslatef()` и заданием значений переноса по координатам `x`, `y` и `z`. Все точки перемещаются вдоль трансформированных осей на заданные значения. Вот пример использования функции `glTranslatef()`.

```
glTranslatef(100.0f, 200.0f, 300.0f);
```

Масштабирование

При масштабировании координаты точек умножаются на заданные значения по осям `x`, `y` и `z`. Например, если точка имеет координаты –5,–2,4, которые умножаются на соответствую-

щие значения 2,–1,3, то новыми координатами точки станут –10,2,12. Вот пример использования функции масштабирования.

```
glScalef(2.0f, 2.0f, 1.0f);
```

Поворот

Результат поворота показан на рис. 26.4.

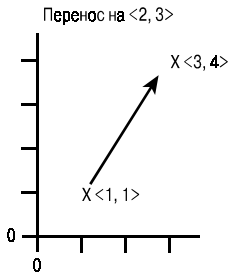


Рис. 26.3. Результат переноса точки

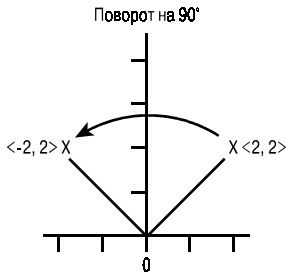


Рис. 26.4. Результат поворота точки

При этом точки совершают поворот вокруг оси, которая определяется относительно их трансформированной локальной координатной оси. Эта ось определяется как вектор, проведенный из начала координат на единичное расстояние в пространстве координат `x`, `y` и `z`. Например, координатное значение 1,0,0 означает положительную часть оси `x`. Для выполнения поворота используйте функцию `glRotatef()`, как показано в следующем примере.

```
glRotatef(45.0f, 1.0f, 0.0f, 0.0f);
glRotatef(180.0f, 0.0f, 1.0f, 0.0f);
glRotatef(23.5f, 0.0f, 0.0f, 1.0f);
```

В результате выполнения этой последовательности изображение будет повернуто на 45,0° вокруг положительной оси x, 180,0° вокруг положительной оси y и на 23,5° вокруг положительной оси z.

Порядок преобразований

Для получения желаемого результата имеет значение, в каком порядке применяются преобразования. Например, важно знать, что результат переноса с последующим поворотом не совпадает с результатом поворота с последующим переносом. Перенос всегда выполняется вдоль локальной оси, а если оси были подвергнуты повороту, то результат переноса будет иным.

Этап 4: рисование примитивов

Рисование примитивов включает задание материала, используемого для примитива, и указание точек в пространстве, определяющих его форму. Этого достаточно для создания сложных и реалистичных объектов.

Установка свойств материалов

В OpenGL материал описывается набором свойств, которые используются для определения характера рисования примитивов. Основные функции обработки и установки свойств материалов входят в набор `glMaterial()`. Их использование показано в листинге 26.5.

Листинг 26.5. Установка свойств материалов

```
// Применив эти цвета, получим отполированный материал красного цвета.
// float AmbientColor[4]={ 0.1f, 0.0f, 0.0f, 1.0f} ,
        DiffuseColor[4]={ 0.8f, 0.0f, 0.0f, 1.0f}
        SpecularColor[4]={ 1.0f, 1.0f, 1.0f, 1.0f} ;
// Устанавливаем для материала "цвет" внешнего света.
glMaterialfv(GL_FRONT_AND_BACK, GL_AMBIENT, AmbientColor);
// Устанавливаем для материала "цвет" рассеянного света.
glMaterialfv(GL_FRONT_AND_BACK, GL_DIFFUSE, DiffuseColor);
// Устанавливаем для материала "цвет" отраженного света.
glMaterialfv(GL_FRONT_AND_BACK, GL_SPECULAR, SpecularColor);
```

Обратите внимание на использование константы GL_FRONT_AND_BACK: вместо нее можно использовать константу GL_FRONT или GL_BACK, что позволяет определять различные материалы для передней и задней сторон примитивов.

Кроме того, используя в качестве второго параметра константу GL_AMBIENT_AND_DIFFUSE, можно установить свойства внешнего и рассеянного света равными одинаковым значениям в одинаковое время.

Существует еще один способ установки материалов для примитивов — он проще, но менее гибок, чем предыдущий метод. Он состоит в использовании функции glColorMaterial(), которая заставляет OpenGL применить материалы, определяемые вызовом функции glColor(). В этом случае цвета материалов будут соответствовать цветам пикселей. Функция glColorMaterial() используется следующим образом.

```
glColorMaterial(GL_FRONT, GL_AMBIENT_AND_DIFFUSE);
// С этого момента свойства материалов, зависящие от внешнего и рассеянного
// света, для передней стороны примитивов будут определяться функцией glColor().
```

После вызова функции glColorMaterial() функцию glColor() можно использовать для определения свойств материалов, как показано в листинге 26.6.

Листинг 26.6. Установка свойств материалов с помощью функций glColorMaterial() и glColor()

```
float ColorArray[4]={ 1.0f, 1.0f, 0.2f, 1.0f} ;

// Устанавливаем текущий цвет, используя отдельные значения.
glColor4f( 1.0f, 0.5f, 0.2f, 1.0f );
// С этого момента к фронтальным сторонам создаваемых примитивов будет применен
// материал, у которого свойства компонентов внешнего и рассеянного освещения
// будут совпадать с цветовыми элементами, используемые ранее.

// Устанавливаем текущий цвет с помощью массива значений.
glColor4fv( ColorArray );
// С этого момента к фронтальным сторонам создаваемых примитивов будет применен
// материал, у которого свойства компонентов внешнего и рассеянного освещения
// будут совпадать со значениями массива, используемыми ранее.
// Для определения порядка точек по часовой стрелке в качестве
// фронтальной стороны:
glFrontFace( GL_CW );
// Для определения порядка точек против часовой стрелке в
```

```
// качестве фронтальной стороны:(это действует по умолчанию):
glFrontFace ( GL_CCW );
```

Можно недоумевать, откуда OpenGL "известно", где фронтальная сторона примитива. Это устанавливается на основе порядка точек, которые определяют примитив (в частности, многоугольник). Точки определяются либо по часовой стрелке, либо против. С помощью команды `glFrontFace()` можно заставить OpenGL считать любую сторону фронтальной.

Каждый режимный флаг OpenGL имеет стандартное значение, которое устанавливается при инициализации OpenGL. Новые значения режимных флагов следует устанавливать только в том случае, если требуется нестандартное поведение OpenGL. Однако из практического опыта стало ясно, что текст программы становится понятнее, если установить все необходимые режимные флаги, не обращая внимание на существование значений, действующих по умолчанию.

Установка нормалей освещения

В математике нормаль — это линия, которая перпендикулярна плоскости. В OpenGL нормаль имеет чуть более гибкое определение. Она по-прежнему привязана к геометрическим примитивам, но не обязательно находится под 90° к ним. В OpenGL нормали используются только для создания освещения. Если вы вспомните составляющие освещения в разделе "Модели освещения и затенения" выше в этой главе (см. рис. 26.2), то сможете припомнить и линию нормали, определяющую, как отражается свет. *Рассеянный* свет отражается под разными углами, и его интенсивность зависит от угла, под которым расположен источник света. Свет, именуемый *отраженным*, отражается только под определенным углом (углом отражения, который равен углу падения света) к нормали. Следовательно, если изменить теоретическую нормаль в модели без изменения реальной плоскости, изменятся и условия освещения.

Теперь представьте определение различных нормалей для различных точек на одной плоской поверхности. В результате получим эффект неплоской поверхности. Именно такие вольности допускаются в OpenGL. Пользователь может определить различные нормали для каждой точки многоугольника, а OpenGL затем интерполирует модель освещения между ними. Такой подход можно использовать для создания эффектов гладкой поверхности в моделях с низкой полигональностью (модель, которая не содержит большого количества многоугольников).

Нормали определяются в OpenGL с помощью набора функций `glNormal()`, как показано в листинге 26.7.

Листинг 26.7. Установка нормали с помощью функции `glNormal()`

```
// Массив вдоль отрицательной оси z.
float NormalArray[3]={ 0.0f, 0.0f, -1.0f} ;

// Устанавливаем нормаль вдоль положительной оси x
// с помощью отдельных значений.
glNormal3f( 1.0f, 0.0f, 0.0f );

// Устанавливаем нормаль с помощью массива значений.
glNormal3fv( NormalArray );
```

Безусловно, для практических целей этот фрагмент программы почти бесполезен, и ваши нормали вряд ли можно будет определить так же просто. Гораздо проще использовать функцию, которая строит нормаль к поверхности на основании трех точек. Это совсем не сложно, если использовать векторный метод, известный как векторное произведение. Что в действительности вычисляет векторное произведение, показано на рис. 26.5.

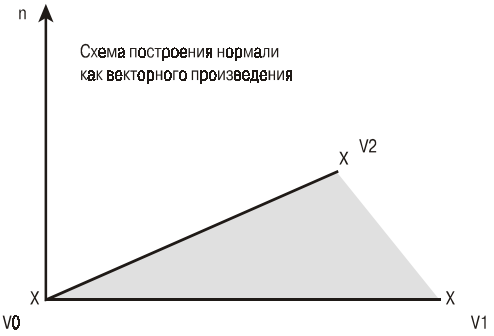


Рис. 26.5. `n` — результат векторного произведения векторов `(v1-v0)` и `(v2-v0)`

Использование функции построения нормали к плоскости показано в листинге 26.8.

Листинг 26.8. Вычисление нормали с помощью векторного произведения

```
#define X       0
#define Y       1
#define Z       2

void WorkOutNormal( float PlanePoint0[3], float PlanePoint1[3],
                    float PlanePoint2[3], float NormalOut[3] )
{
    float VectorA[3], VectorB[3];
    double Length, OneOverLength;

    // Вычисляем два вектора, принадлежащие плоскости и выходящие
    // из одной точки.
    VectorA[X] = PlanePoint1[X] - PlanePoint0[X];
    VectorA[Y] = PlanePoint1[Y] - PlanePoint0[Y];
    VectorA[Z] = PlanePoint1[Z] - PlanePoint0[Z];
    VectorB[X] = PlanePoint2[X] - PlanePoint0[X];
    VectorB[Y] = PlanePoint2[Y] - PlanePoint0[Y];
    VectorB[Z] = PlanePoint2[Z] - PlanePoint0[Z];

    // Вычисляем векторное произведение.
    NormalOut[X] = VectorA[Y] * VectorB[Z] - VectorA[Z] * VectorB[Y];
    NormalOut[X] = VectorA[Z] * VectorB[X] - VectorA[X] * VectorB[Z];
    NormalOut[X] = VectorA[X] * VectorB[Y] - VectorA[Y] * VectorB[X];

    // Сокращаем нормаль до единичного вектора.
    Length = NormalOut[X] * NormalOut[X] + NormalOut[Y] *
                     NormalOut[Y] + NormalOut[Z] * NormalOut[Z];
```

```
    // Нужно убедиться, что квадратный корень действителен.
    if( Length > 0 )
        OneOverLength = 1/sqrt( Length );
    else
        OneOverLength = (1/0.0001);

    NormalOut[X] *= OneOverLength;
    NormalOut[Y] *= OneOverLength;
    NormalOut[Z] *= OneOverLength;
}
```

Использование функции, определенной в листинге 26.8, показано в листинге 26.9.

Листинг 26.9. Создание и использование нормали с помощью функции `WorkOutNormal()`

```
float Vertex0[3]={ -1.0f, -1.0f, 0.0f} ,
Vertex1[3]={ 0.0f, 1.0f, 0.0f} ,
Vertex2[3]={ 1.0f, -1.0f, 0.0f} ;
float Normal[3];

WorkOutNormal(Vertex0, Vertex1, Vertex2, Normal);
// Теперь массив Normal содержит математическую нормаль к
// плоскости, определяемую векторами Vertex0, Vertex1 и Vertex2.
glNormal3fv(Normal);
```

Безусловно, функция `WorkOutNormal()` строит только математически строгую нормаль к поверхности, что не позволяет использовать преимущества OpenGL, заключающиеся в возможности использовать произвольные нормали. Существует несколько методов, которые можно применить к созданию нормалей в целях имитации гладких поверхностей. Большинство из них включает усреднение нормалей близлежащих многоугольников, но их описание выходит за рамки этой главы.

Выполнение операций рисования

В следующих разделах раскрываются детали спецификации реальной сцены. Сюда входит задание типа создаваемого примитива и способа его рисования. Под примитивом понимается базовая фигура, например точка, линия и заполненный треугольник.

В OpenGL команды рисования выполняются не сразу же после их выдачи, а "становятся" в очередь, которая обрабатывается, когда OpenGL получает на это время, либо при вызове функции `glFlush()`.

Порядок команд в очереди всегда сохраняется одинаковым.

Определение примитивов

Все примитивы в OpenGL (точки, линии, многоугольники и различные полосы) определяются с помощью точек. Точка характеризуется местоположением (позицией) в трехмерном пространстве, обычно определяемой тремя числами: координатами `x`, `y` и `z`. Для их задания в

OpenGL существует набор команд `glVertex()`, предусматривающий различные типы данных. Среди них есть функции, принимающие только два параметра. Они предназначены для рисования псевдодвумерных примитивов, в которых координата `z` принимается равной `0`.

Тип примитива, описываемого некоторым количеством вершин (трехмерных или двухмерных точек), определяется параметром, переданным функции `glBegin()`. Вершины примитива задаются с помощью функций `glVertex()` между обращениями к функциям `glBegin()` и `glEnd()`.

```
// Рисуем треугольник.
glBegin( GL_TRIANGLES );
    glVertex3f( -10.0f, -10.0f, 0.0f );
    glVertex3f( 0.0f, 10.0f, 0.0f );
    glVertex3f( 10.0f, -10.0f, 0.0f );
glEnd();
```

Функции `glBegin()` и `glEnd()` можно вызывать в любой точке программы при выполнении следующих условий.

- Эти вызовы не должны быть вложенными (т.е. пара функций `glBegin()`/`glEnd()` не вызывается внутри другой пары функций `glBegin()`/`glEnd()`).
- Интерфейс OpenGL должен быть инициализирован, т.е. созданы контексты рендеринга и устройства.
- Каждая функция `glBegin()` должна иметь соответствующую функцию `glEnd()`.

Многоугольники, точки и линии

Многоугольник (или полигон) — это примитив, определяемый более чем двумя точками. Он обладает площадью поверхности. *Точкой* является позиция в трехмерном пространстве. Она не имеет площади поверхности. *Линия* соединяет между собой две точки. Она также не имеет площади поверхности. Эти (и другие) примитивы OpenGL (с соответствующими им константами) перечислены в табл. 26.1.

Таблица 26.1. Примитивы OpenGL и их константы

Константа	Примитив
`GL_POINT`	Вершина или точка
`GL_LINE`	Линия
`GL_TRIANGLE`	Трехсторонний многоугольник, определяемый тремя точками
`GL_QUAD`	Четырехсторонний многоугольник, определяемый четырьмя точками
`GL_POLYGON`	*n*-сторонний многоугольник, определяемый *n* точками между функциями `glBegin()` и `glEnd()`
`GL_TRIANGLE_STRIP`	*n* треугольников в полосе, определяемой *n*+2 точками
`GL_QUAD_STRIP`	*n* четырехугольников в полосе, определяемой 2*n*+2 точками

Рисование примитивов с помощью OpenGL

Примитивы очень легко рисовать с помощью OpenGL. Они всегда определяются в соответствии с локальной системой координат. (Координатные системы рассматриваются ниже в этой главе.) В листинге 26.10 приведен пример рисования куба красного цвета с нормалями для создания освещения.

Листинг 26.10. Рисование объекта с использованием цвета и нормалей

```
// Примечание: не забудьте установить до начала рисования
// OpenGL-контексты и свойства материалов.

// Устанавливаем цвет вершины равным красному.
glColor3f( 1.0f, 0.0f, 0.0f );

// Не забудьте сообщить OpenGL тип определяемого примитива.
// Мы рисуем квадраты (четырехсторонние многоугольники).
glBegin( GL_QUADS );
// Передняя грань.
glNormalf( 0.0f, 0.0f, 1.0f );
glVertex3f( -1.0f, 1.0f, -1.0f );
glVertex3f( 1.0f, 1.0f, -1.0f );
glVertex3f( 1.0f, -1.0f, -1.0f );
glVertex3f( -1.0f, -1.0f, -1.0f );
// Задняя грань.
glNormalf( 0.0f, 0.0f, -1.0f );
glVertex3f( 1.0f, 1.0f, 1.0f );
glVertex3f( -1.0f, 1.0f, 1.0f );
glVertex3f( -1.0f, -1.0f, 1.0f );
glVertex3f( 1.0f, -1.0f, 1.0f );
// Правая грань.
glNormalf( 1.0f, 0.0f, 0.0f );
glVertex3f( 1.0f, 1.0f, -1.0f );
glVertex3f( 1.0f, 1.0f, 1.0f );
glVertex3f( 1.0f, -1.0f, 1.0f );
glVertex3f( 1.0f, -1.0f, -1.0f );
// Левая грань.
glNormalf( -1.0f, 0.0f, 0.0f );
glVertex3f( -1.0f, 1.0f, -1.0f )
glVertex3f( -1.0f, 1.0f, 1.0f );
glVertex3f( -1.0f, -1.0f, 1.0f );
glVertex3f( -1.0f, -1.0f, -1.0f );
// Верхняя грань.
glNormalf( 0.0f, 1.0f, 0.0f );
glVertex3f( -1.0f, 1.0f, 1.0f );
glVertex3f( 1.0f, 1.0f, 1.0f );
glVertex3f( 1.0f, 1.0f, -1.0f );
glVertex3f( -1.0f, 1.0f, -1.0f );
// Нижняя грань.
glNormalf( 0.0f, -1.0f, 0.0f );
glVertex3f( -1.0f, -1.0f,  -1.0f );
glVertex3f( 1.0f, -1.0f, -1.0f );
glVertex3f( 1.0f, -1.0f, 1.0f );
glVertex3f( -1.0f, -1.0f, 1.0f );
glEnd();
// Даем сигнал выполнения команд рисования из очереди.
glFlush();
```

Нетрудно заметить, что многие определения вершин повторяются. Очевидно, что это неэффективно, поэтому в OpenGL предусмотрены собственные методы для "борьбы" с такого рода неэффективностью (`glVertexArrays()` и пр.), но они здесь не описываются. Мы же рассмотрим другой метод, состоящий в сохранении определений вершин, примитивов и нормалей в массивах. Этот способ построения куба показан в листинге 26.11.

Листинг 26.11. Рисование объекта с использованием массива вершин и многоугольников

```
// Сначала определяем массивы.
// Массив вершин:
float Verts[8][3]=
{
    { -1.0f, 1.0f, -1.0f} ,
    { 1.0f, 1.0f, -1.0f} ,
    { 1.0f, -1.0f, -1.0f} ,
    { -1.0f, -1.0f, -1.0f} ,
    { 1.0f, 1.0f, 1.0f} ,
    { -1.0f, 1.0f, 1.0f} ,
    { -1.0f, -1.0f, 1.0f} ,
    { 1.0f, -1.0f, 1.0f}
} ;
// Массив примитивов, содержащий индексы для массива Verts,
// который определяет квадраты.
int Polys[6][3]=
{
    { 0, 1, 2, 3} ,
    { 4, 5, 7, 6} ,
    { 1, 4, 7, 2} ,
    { 5, 0, 3, 6} ,
    { 5, 4, 1, 0} ,
    { 3, 2, 7, 6}
} ;
// Нормали для квадратов.
float Normals[6][3]=
{
    { 0.0f, 0.0f, -1.0f} ,
    { 0.0f, 0.0f, 1.0f}
    { 1.0f, 0.0f, 0.0f} ,
    { -1.0f, 0.0f, 0.0f} ,
    { 1.0f, 0.0f, 0.0f} ,
    { -1.0f, 0.0f, 0.0f}
} ;

// Теперь можно нарисовать куб в цикле:
int i;

glColor3f(1.0f, 0.0f,  0.0f);
glBegin(GL_QUADS);
    for(i=0; i<6; i++)
    {
        glNormalfv(Normals[i]);
```

```
        glVertex3fv(Verts[Polys[i][0]]);
        glVertex3fv(Verts[Polys[i][1]]);
        glVertex3fv(Verts[Polys[i][2]]);
    }
glEnd();
// Активизируем процесс выполнения команд из очереди.
glFlush();
```

С точки зрения программирования, это гораздо более эффективный способ рисования куба, который представляет собой объект, составленный из определений вершин и примитивов. К сожалению, с точки зрения обработки кода, этот метод нельзя назвать эффективным, если речь идет о создании нескольких фигур.

Списки отображения (более эффективный способ создания примитивов)

OpenGL позволяет применить более компактный (с точки зрения программирования) метод определения спецификации сцены и манипуляции ее содержимым, по сравнению с методами, описанными в предыдущем разделе. Списки отображения по сути представляют собой макросы рисования. Они содержат команды OpenGL, которые задает программист. Такие списки можно затем выполнить в любой точке программы с помощью одной команды.

Преимущества использования списков отображения заключаются в большей эффективности их выполнения, по сравнению с выполнением отдельных команд. Кроме того, такой подход способствует соблюдению четкого стиля программирования.

Каждый список, создаваемый программистом в OpenGL, использует идентификационный номер (ID), который обеспечивается либо самостоятельно, либо запрашивается у OpenGL. Второй вариант предпочтительнее, поскольку освобождает программиста от необходимости отслеживать свободные номера ID. Поэтому первый шаг в создании списка отображения — запросить свободный номер ID с помощью функции `glGenLists()`. Эта функция вызывается с целочисленным параметром, который задает количество свободных номеров (ID) списков, запрашиваемых для выделения. Она возвращает первый выделенный ID. Остальные ID располагаются последовательно за первым (первый ID, первый ID+1, первый ID+2 и т.д.).

Следующий этап — регистрация списка с помощью команды `glNewList()`. Первый параметр функции `glNewList()` — номер ID списка. Второй — константа, которая предписывает OpenGL запомнить список для использования в будущем или зарегистрировать его и выполнить. Для этих двух вариантов поведения предусмотрены константы `GL_COMPILE` и `GL_COMPILE_AND_EXECUTE`.

Команда OpenGL (за некоторыми исключениями — см. спецификации OpenGL), выполняемая между вызовами функций `glNewList()` и `glEndList()`, скомпилируется в список.

Чтобы выполнить список команд, достаточно вызвать функцию `glCallList( ID )`. Пример приведен в листинге 26.12.

Листинг 26.12. Создание списка отображения

```
int ListID;

// Получаем свободный номер ID списка.
ListID = glGenLists( 1 );

// Начинаем компиляцию списка.
glNewList( ListID, GL_COMPILE );
```

```
// Здесь должны быть команды OpenGL.

// Прекращаем компиляцию списка.
glEndList();

// Выполняем команды ...

// Вызываем список.
glCallList( ListID );
```

Этап 5: Всплытие на поверхность

Этап 5 состоит из вызова одной функции `SwapBuffers()`. Ее использование демонстрируется в функции `UpdateOpenGLScene()` OpenGL-программы, описанной в следующем разделе.

Пример OpenGL-программы

Пример программы, в которой демонстрируются ранее рассмотренные идеи OpenGL, можно найти в папке `OpenGL` на прилагаемом к книге компакт-диске. Файл проекта называется `OpenGLExample.bpr`. Эта программа имитирует солнечную систему. Модель солнечной системы строится иерархически, начиная от центральной звезды, а затем переходя к каждой планете, вращающейся вокруг центра по своей орбите, и к спутникам каждой планеты (такая иерархия теоретически может "опуститься" на бесконечную глубину орбитальных систем). С каждым спутником (так называются тела, вращающиеся по своим орбитам) связываются следующие параметры:

- орбитальная скорость (измеряется в градусах на единицу времени);
- текущая позиция на орбите (измеряется в градусах);
- орбитальное расстояние от объекта, вокруг которого происходит вращение (измеряется в единицах длины);
- радиус реального спутника (измеряется в единицах длины);
- цвет спутника (измеряется в цветовых составляющих: красного, зеленого, синего и альфа);
- квадратичный объект (Quadric Object), который используется в OpenGL для рисования спутника;
- спутники, вращающиеся вокруг планеты и представляющие собой следующий уровень иерархии.

Используемые в программе методы делают ее очень простой. Прежде всего устанавливаются контексты устройства и рендеринга OpenGL. Затем в начало координат устанавливается источник света для создания эффекта звезды. Следующий этап — инициализация модели солнечной системы. Вам осталось лишь добавить спутники. Эта модель не вполне пропорциональна, но если вы хотите почувствовать "бесконечность" расстояний между нашими планетами, попробуйте установить константы `DISTANCE_DIV`, `SUN_RADIUS_DIV` и `RADIUS_DIV` равными числу `500000.0f` и перекомпилируйте программу.

Все этапы рисования выполняются в обработчике событий `OnIdle()`, который настроен на выполнение функции `UpdateOpenGLScene()`. Ниже перечислены действия функции `UpdateOpenGLScene()`.

- Очистка экрана.
- Очистка матрицы преобразования.
- Преобразование относительно расстояния до наблюдения с учетом поворота.
- Вызов функции рисования звезды.
- Очистка буфера глубины (чтобы все рисование выполнялось поверх звезд).
- Вызов функции рисования солнечной системы (необходимо вызвать только функцию для солнца; все остальные вызываются иерархически из нее).
- Вызов функции рисования орбит в солнечной системе.
- Выполнение команд рисования из очереди OpenGL.
- Смена буфера при двойной буферизации.

Для регулярного обновления положения спутников устанавливается таймер. Причем это делается только в том случае, если разрешен режим анимации, который управляется кнопкой Play. Регулятор скорости (понятное дело) предназначен для регулировки скорости анимации. Комментарии в исходном тексте программы помогут понять, что в действительности происходит на каждом этапе создания сцены.

Эта программа была разработана и написана для 24-разрядного цветового графического режима, поэтому такая настройка была бы идеальной для работы этой программы. В любом другом режиме (в том числе и в 256-цветном) эта OpenGL-программа не сгенерирует корректного изображения.

Этот проект должен также содержать две библиотеки `OpenGL32.lib` и `Glu32.lib`. Они должны находиться в стандартной папке C++Builder `LIB\PSDK`.

В процессе работы программы левую кнопку мыши можно использовать для поворота изображения (удерживая нажатой, перетаскивайте мышь), правую кнопку — для увеличения (уменьшения) масштаба изображения и обе кнопки — для панорамирования изображения.

Резюме по OpenGL

Программирование с использованием OpenGL API — огромная тема, которой посвящены сотни страниц многих книг. В этой главе я попытался затронуть самые основные и наиболее важные ее аспекты, чтобы указать читателю путь к созданию высококачественной трехмерной графики в режиме реального времени. По понятным причинам некоторые разделы этого API были совершенно опущены (например, отображение текстур), но изложенного вполне достаточно для разработки приложений с использованием OpenGL.

Дополнительные источники информации по OpenGL

Если вы хотите узнать больше об интерфейсе OpenGL и программировании трехмерных графических изображений, обратитесь к следующим источникам.

- Web-сайт OpenGL по адресу: `http://www.opengl.org`.

  Новинки технологии, обучение и пр.

- Список почтовой рассылки OpenGL (отправьте электронной почтой свою заявку по адресу: `ListGuru@fatcity.com` с указанием темы `SUBSCRIBE OPENGL-GAMEDEV-L`).

  Сюда я обращаюсь, если у меня есть вопросы (технического или иного характера) и всегда быстро получаю ответ.

- *OpenGL Programming Guide* (“Красная книга”) второе издание, OpenGL Architecture Review Board (Mason Woo, Jackie Neider, Tom Davis), ISBN 0-201-46138-2, Addison-Wesley.

Исчерпывающее руководство по OpenGL, написанное его разработчиками. Для начинающих несколько сложно, но это самая полная книга из тех, что мне попадались по этой теме.

- *OpenGL SuperBible* (“Голубая книга”), Richard S. Wright Jr., Michael Sweet, ISBN 1-57169-073-5, Waite Group Press.

  Это более доступная для читателя книга, чем *OpenGL Programming Guide*. Она посвящена, в частности, Microsoft-реализации OpenGL, ориентирована на платформу Windows и сопровождается прекрасным компакт-диском с примерами.

Введение в DirectX

DirectX — это высокопроизводительный интерфейс с мультимедиа-аппаратурой (разработка компании Microsoft). Его название (Direct — означает *прямой*) говорит о том, что этот API-интерфейс предоставляет программисту прямое взаимодействие с драйверами низкого уровня. Интерфейс DirectX был задуман для программирования компьютерных игр, но многие программисты различных приложений используют его для работы с анимационными и аудио-ресурсами, которые требуют от системы высокой производительности. DirectX предназначен для работы только в операционных системах компании Microsoft (в частности, Windows 9x, 2000 и NT).

COM-ориентация интерфейса DirectX API

Архитектура DirectX опирается на модель компонентных объектов (Component Object Model — COM). COM — это метод программирования, позволяющий приложениям использовать так называемый COM-объект (подобный классу C++), который может быть параллельно использован другими приложениями. Все COM-объекты опираются на один и тот же класс `IUnknown`, содержащий следующие три функции.

- Функция `AddRef()` инкрементирует счетчик внутренних ссылок объекта (содержащий количество других программ, которые используют этот объект).
- Функция `Release()` декрементирует счетчик внутренних ссылок объекта.
- Функция `QueryInterface()` используется для получения интерфейса с объектом, который делает возможным его использование.

На заметку Более подробно о модели компонентных объектов можно прочитать в главе 16.

Используя DirectX, вы можете не беспокоиться о работе спецификации COM, поскольку функции DirectX API практически так же просты, как и объекты COM.

Необъектные функции DirectX

В DirectX предусмотрено несколько функций, которые не являются частью какого бы то ни было из существующих API-интерфейсов или объектов. В большинстве это функции инициализации. Вам также придется использовать некоторые специальные COM-функции, а именно:

- функцию `CoInitialize()`, которая подготавливает программу к использованию COM-объектов;
- функцию `CoCreateInstance()`, которая создает экземпляр конкретного COM-объекта.

Использование обеих функций показано в листинге 26.14.

Использование DirectDraw

DirectDraw — это API DirectX, ориентированный на поддержку двумерной растровой графики. Он гораздо быстрее и мощнее, чем интерфейс Windows GDI. Этот раздел можно рассматривать как знакомство с самыми полезными его функциями.

Инициализация объекта DirectDraw

Объекты DirectDraw создаются с помощью функции `DirectDrawCreate()`. Этой функции следует передать GUID устройства и указатель на объект DirectDraw. GUID устройства представляет собой номер, означающий аппаратное устройство, посредством которого вы собираетесь рисовать. Для выбора первичного устройства (обычно это главный графический адаптер) можно в качестве этого параметра передать значение `NULL`. Существует также возможность узнать состав всех подключенных и зарегистрированных в компьютере устройств. Такой учет аппаратных ресурсов можно выполнить с помощью функции `DirectDrawCreate()`. Мы не будем останавливаться здесь на описании этого метода, но он довольно прост и хорошо описан в другой литературе и файлах справочной системы.

Чтобы благополучно создать объект DirectDraw, можно воспользоваться вариантом, продемонстрированным в листинге 26.13.

Листинг 26.13. Создание объекта DirectDraw

```
// Стандартным типом значений, возвращаемых функциями DirectX,
// является HRESULT. Возвращаемое значение содержит либо константу DD_OK,
// если функция успешно выполнилась, либо код ошибки.
// HRESULT DDError;
// Это указатель на реальный объект DirectDraw.
LPDIRECTDRAW tempDD;

// Попробуем получить DD-объект (от термина DirectDraw).
DDError=DirectDrawCreate(NULL, &tempDD, NULL);
// Если произошла ошибка (проверяем с помощью макроса FAILED)...
if(FAILED(DDError))
{
    // ...возвращаем признак ошибки.
    return ( false );
}
```

Этого достаточно для создания стандартного интерфейса DirectDraw. Причем мы обошлись без COM-объектов. COM потребуется в том случае, если мы захотим получить сопряжение с более новой версией интерфейса DirectDraw. На момент написания этих строк самой свежей была версия 7.0, а 8.0 — "на подходе". С помощью представленных выше методов мы получаем только базовый объект DirectDraw. Чтобы использовать

другую версию, после создания объекта DirectDraw (по примеру листинга 26.13) действуйте, как показано в листинге 26.14.

Листинг 26.14. Получение другой версии интерфейса DirectDraw

```
// Интерфейс в DirectDraw2
LPDIRECTDRAW2 DirectDraw2Interface;

// Проверяем наличие DirectDraw2 в системе.
DDError=tempDD->QueryInterface(IID_IDirectDraw2,
                               (LPVOID *)&DirectDraw2Interface);

// Если мы не можем найти его, этот макрос возвратит признак ошибки.
// if(FAILED(DDError))
{
    return ( false );
}
// Подготовка к созданию COM-объекта.
CoInitialize(NULL);

// Пытаемся создать интерфейс DD2.
DDError=CoCreateInstance(CLSID_DirectDraw, NULL, CLSCTX_ALL,
              IID_IDirectDraw2, (LPVOID *)&DirectDraw2Interface);

// В случае неудачи будет возвращен признак ошибки.
if(FAILED(DDError))
{
    return ( false );
}
// Теперь попробуем инициализировать объект DirectDraw,
// используя стандартное устройство (NULL).
DDError=DirectDraw2Interface->Initialize(NULL);
if(FAILED(DDError))
{
    return ( false );
}
// Теперь можно отключить инициализацию COM-объекта.
CoUninitialize();

// Возвращаем признак успешного выполнения.
return  ( true );
```

В этой программе делается попытка обнаружить интерфейс DirectDraw 2, и в случае его отсутствия возвращается признак ошибки. Лучше всего использовать по возможности самый новый интерфейс, и поэтому в программе следует предусмотреть соответствующие настройки. Функциональные возможности, описанные в этой главе, соответствуют базовому уровню, который включен в каждую версию DirectX, поэтому в целях изучения не стоит беспокоиться о том, чтобы достать самую последнюю его версию (хотя на практике это, безусловно, приветствуется).

Настройка параметров отображения для DirectDraw

DirectDraw позволяет программисту полностью забыть об ограничениях рабочего стола и действовать по своему усмотрению, например разрешить пользователю изменить режим отображения и перейти к полному экрану.

Прежде чем изменять разрешение экрана (его ширину и длину в пикселях), необходимо установить параметр, именуемый *кооперативным уровнем*. Он определяет, как Windows "относится" к данному приложению. Если кооперативный уровень включает флаг `DDSCL_EXCLUSIVE`, Windows предоставляет этому приложению полный контроль над областью отображения. Для установки полноэкранного режима для приложения этот флаг используйте в конъюнкции с флагом `DDSCL_FULLSCREEN`. Если используется флаг `DDSCL_NORMAL`, то приложение будет выполняться в окне. Этот флаг нельзя использовать совместно с флагом `DDSCL_EXCLUSIVE`. Существуют и другие, менее важные флаги.

В листинге 26.15 показано, как перейти в полноэкранный режим и пользоваться монопольным управлением областью отображения.

Листинг 26.15. Установка полноэкранного режима и монопольного управления областью отображения

```
bool SetExclusiveMode( HWND Window )
{
    HRESULT DDError;

if(DirectDraw2Interface==NULL)
        return(false);
    // Попытка установить полноэкранный режим с монопольным управлением.
    // DDError = DirectDraw2Interface->SetCooperativeLevel( Window,
                                          DDSCL_ALLOWREBOOT |
                                          DDSCL_EXCLUSIVE |
                                          DDSCL_FULLSCREEN );
    // Если возникла ошибка...
    if(DDError!=DD_OK)
    {
        return(false);
    }
    return(true);
}
```

На заметку Флаг `DDSCL_ALLOWREBOOT` в функции `SetCooperativeLevel()` разрешает пользователю перезагрузить компьютер с помощью клавиш <Alt+Ctrl+Del>.

Используя интерфейс DirectDraw, можно также изменить разрешение устройства отображения (видеомонитора и карты). Это реализуется с помощью функции `SetDisplayMode()` интерфейса DirectDraw.

```
DirectDraw2Interface->SetDisplayMode( Width, Height,
                                BitsPerPixel, RefreshRate, Flags);
```

Параметры этой функции и их значения приведены в табл. 26.2.

Таблица 26.2. Параметры функции `SetDisplayMode()`

Имя параметра	Описание	Единицы
`Width`	Желаемая ширина экрана	Пиксели
`Height`	Желаемая высота экрана	Пиксели
`BitsPerPixel`	Желаемая разрядность пикселей	Биты на пиксель
`RefreshRate`	Желаемая скорость обновления монитора; используйте 0, если это вам безразлично	Герцы (число обновлений в секунду)
`Flags`	Дополнительная информация	-

Чтобы установить режим отображения с разрешением 640×480 и 16-разрядным цветом, используйте функцию `SetDisplayMode()`.

```
DirectDraw2Interface->SetDisplayMode(640, 480, 16, 0, 0);
```

Большинство (если не все) современных графических карт позволяют установить этот графический режим. Это стандартное разрешение для компьютерных игр, рассчитанных на высокую производительность системы. При выходе из приложения будет автоматически восстановлен прежний режим отображения, поэтому вам самим не нужно беспокоиться на этот счет.

На заметку Используя DirectDraw, можно еще до установки определенного режима отображения получить полный список всех возможных режимов, обеспечиваемых используемой графической картой. Если вы установите режим, имеющийся в этом списке, то шансы получить нужный режим значительно возрастут. К сожалению, методы, используемые для получения такого списка, выходят за рамки настоящей главы.

Поверхности рисования

Объект DirectDraw доказал свою пользу в описанных выше методах исключительно при подготовке компьютера и видеокарты к работе в высокоскоростных графических режимах. Чтобы реально нарисовать что-либо на экране, необходимо использовать нечто, именуемое *поверхностью* (surface). Для создания и манипулирования такими объектами программист использует класс `IDirectDrawSurface`. Поверхность — это "рисовальное" полотно (канва), подобное дисплею, но содержащееся в памяти. По своей роли оно аналогично классу `TCanvas`, но работает почти в 1000 раз быстрее (по грубым оценкам).

Поверхности, создаваемые с помощью объекта DirectDraw, уже "готовы к употреблению", т.е. инициализированы. Прежде всего нужно создать *первичную поверхность*. Это — главная поверхность рисования, которая в конце концов станет поверхностью экрана. Эта поверхность может быть буферизирована либо дважды, либо единожды.

На заметку Двойная буферизация означает, что для отображения реально используется две поверхности. Одна из них отображается на экране, а содержимое другой заменяется. После внесения изменений они меняются местами. Преимущество этого метода в том, что пользователь не замечает обновления изображения на экране — он видит только один кадр. Единичная буферизация означает, что внесение изменений в изображение производится прямо в видимый буфер. Конечно, это не добавляет изображению привлекательности, особенно в случае, если очередной кадр содержит пустой экран.

В листинге 26.16 представлен фрагмент программы, демонстрирующий использование метода анимации с двойной буферизацией для создания первичного буфера.

Листинг 26.16. Создание первичного буфера DirectDraw

```
// Переменные объявлены глобально или в классе.
LPDIRECTDRAWSURFACE3 pPrimarySurface, pBackBuffer;

bool CreateSurfaces( void )
{
    HRESULT DDError;
    // Эти определения используются для описания типа создаваемой поверхности.
    DDSURFACEDESC SurfaceDescription;
    DDSCAPS BackBufferCaps;

    if(DirectDraw2Interface==NULL)
        return(false);

    // Обнуляем все описание.
    ZeroMemory(&SurfaceDescription, sizeof(DDSURFACEDESC));
    // Всегда нужно устанавливать член dwSize структур DirectX
    // равным размеру структуры.
    SurfaceDescription.dwSize = sizeof(DDSURFACEDESC);
    // Указываем для процедуры создания, какие поля в описании
    // являются действительными.
    SurfaceDescription.dwFlags = DDSD_CAPS | DDSD_BACKBUFFERCOUNT;
    // Устанавливаем свойства, которые мы хотели бы получить.

    SurfaceDescription.ddsCaps.dwCaps =
        // Первичная поверхность всегда отображается.
        DDSCAPS_PRIMARYSURFACE |
        // Она будет дважды буферизирована.
        DDSCAPS_FLIP |
        // С ней связано больше поверхностей (т.е. еще один
        // буфер, благодаря которому она дважды буферизирована).
        DDSCAPS_COMPLEX;
    // Мы хотим, чтобы 1 буфер был связан с первичной
    // поверхностью (двойной буфер).
    SurfaceDescription.dwBackBufferCount = 1;

    //Пытаемся создать поверхность.
    DDError = DirectDraw2Interface->CreateSurface(
                  &SurfaceDescription, (LPDIRECTDRAWSURFACE *)
                  &pPrimarySurface, NULL);
    // В случае неудачи...
    if(FAILED(DDError))
    {
        // ...возвращаем признак ошибки.
        return(false);
    }
    // Обнуляем описание заднего буфера.
    ZeroMemory(&BackBufferCaps, sizeof(DDSCAPS));
    // Предписываем создание заднего буфера (backbuffer).
    BackBufferCaps.dwCaps = DDSCAPS_BACKBUFFER;
```

```
    // Пытаемся получить его из первичной поверхности.
    DDError = pPrimarySurface->GetAttachedSurface(&BackBufferCaps, &pBackBuffer);
    // Если создание не удается, возвращаем признак ошибки после
    // освобождения первичной поверхности.
    if(FAILED(DDError))
    {
        // Вызываем функцию освобождения для удаления первичной поверхности.
        // pPrimarySurface ->Release();
        return(false);
    }
    return (true);
}
```

Использование GDI на поверхностях DirectDraw

Для упрощения "графических работ" на поверхности DirectDraw можно использовать интерфейс с графическими устройствами Windows (graphics device interface — GDI). Для этого достаточно получить контекст устройства для поверхности, нарисовать задуманное, а затем освободить занятый контекст устройства.

Чтобы получить контекст устройства, можно использовать функцию `GetDC()` интерфейса `IDirectDrawSurface`. Затем на этом контексте устройства можно использовать стандартные GDI-функции. По окончании рисования обязательно нужно вызвать функцию `ReleaseDC()`, чтобы освободить ранее полученный контекст устройства.

В листинге 26.17 показано, как записать текст на поверхность с помощью GDI.

Листинг 26.17. Запись текста на поверхность с помощью GDI

```
// Объявляем переменные глобально или в классе.
LPDIRECTDRAWSURFACE3 pPrimarySurface, pBackBuffer;

bool TextOutDD( int x, int y, char *szText, COLORREF color,
                COLORREF backcolor)
{
    // Контекст устройства.
    HDC hDeviceContext;
    // Эта переменная будет хранить значение ошибки (на случай,
    // если таковая произойдет).
    HRESULT DDError;

    if(pBackBuffer==NULL)
        return(false);
    // Пробуем получить контекст устройства для заднего буфера.
    DDError = pBackBuffer->GetDC(&hDeviceContext);

    // Если это нам не удалось...
    if( FAILED( DDError ) )
    {
        // ...возвращаем признак ошибки.
        return( false );
    }
```

```
    // Устанавливаем для текста цвет фона.
    SetBkColor(hDeviceContext, backcolor);
    // Устанавливаем для текста основной цвет.
    SetTextColor(hDeviceContext, color);

    // Выводим текст.
    ::TextOut(hDeviceContext, x, y, szText, lstrlen(szText));

    // Освобождаем контекст устройства для поверхности.
    pBackBuffer->ReleaseDC(hDeviceContext);
    // Возвращаем признак отсутствия ошибки.
    return( true );
}
```

В листинге 26.17 рисование происходит в заднем буфере (`pBackBuffer`). Все это прекрасно, но результат рисования не отображается на экране. Чтобы поменять буферы местами, используйте функцию `Flip()` интерфейса `IDirectDrawSurface`. Но если буфер окажется потерянным (иногда это случается), такая ситуация будет отмечена значением ошибки, возвращаемым функцией `Flip()`. Эту ошибку можно легко перехватить и исправить путем повторной инициализации поверхностей. Функция, представленная в листинге 26.18, выполняет надежную подкачку данных за счет контроля ошибок.

Листинг 26.18. Переключение дважды буферизированных поверхностей

```
// Переменные, объявленные глобально или в классе.
LPDIRECTDRAWSURFACE3 pPrimarySurface, pBackBuffer;

void FlipSurfaces( void )
{
    HRESULT DDError;

    // Работаем в цикле до тех пор, пока не будет возвращено ни одной ошибки,
    // или DirectDraw сообщит, что рисование все еще продолжается.
    // while( true )
    {
        // Пытаемся поменять местами (переключить) поверхности.
        DDError = pPrimarySurface->Flip(NULL, 0);
        // Если нам это удалось, выходим.
        if( !FAILED( DDError ) )
        {
            break;
        }
        // В противном случае определяем, что ошибочно.
        else if( DDError == DDERR_SURFACELOST )
        {
            // Поверхности были утеряны, поэтому выполняем их
            // повторную инициализацию.
            CreateSurfaces();
            // Вызываем функцию восстановления, чтобы вновь обрести изображение.
            DDError = pPrimarySurface->Restore();
            // Если это не удалось, лучше выйти, поскольку тут уж не до шуток.
```

```
            if( FAILED(DDError) )
                break;
        }
        // Если DirectDraw еще рисовал, когда мы пытались выполнить обмен
        // буферов, следует попробовать реализовать обмен до тех пор,
        // пока он не состоится. В противном случае...
        else if( DDError != DDERR_WASSTILLDRAWING )
        {
            // ...мы выйдем, и вскоре снова выполним обмен.
            break;
        }
    }
}
```

Загрузка растровых изображений на поверхность

Конечно, это здорово — иметь возможность выводить на поверхность текст и рисовать линии, но куда полезнее было бы умение создавать растровые изображения. Это можно легко осуществить с помощью компонента `TBitmap`. Сначала необходимо загрузить соответствующий файл (обычно он имеет расширение `BMP`). В листинге 26.19 приведен фрагмент программы, в котором показано, как загрузить растровый файл в компонент `TBitmap`.

Листинг 26.19. Загрузка BMP-файла в компонент `TBitmap`

```
bool LoadBitmapIntoTBitmap( Graphics::TBitmap *&NewBitmap, AnsiString FileName )
{
    // Присвоим значение NULL экземпляру компонента TBitmap,
    // чтобы можно было сообщить об успешном выделении памяти.
    NewBitmap = NULL;
    // Делаем запрос на выделение памяти.
    NewBitmap = new Graphics::TBitmap();
    // Загружаем растровый файл.
    NewBitmap->LoadFromFile( FileName );

    if(NewBitmap==NULL)
        return( false );
    // Возвращаем признак отсутствия ошибки.
    return(true);
}
```

Как видите, переменную типа `TBitmap` нужно передавать функции `LoadBitmapIntoTBitmap()` в "свободном" состоянии, поскольку выделение памяти осуществляется в самой функции. Если функция возвращает значение `true`, значит, ее работа была успешной, и растровое содержимое заданного файла связано со свойством `Canvas` компонента `TBitmap`. Эту функцию можно легко модифицировать для загрузки растрового изображения из ресурсной составляющей (ID) в выполняемом файле. Для этого замените задание параметра `AnsiString FileName` сочетанием `int ResourceID`, а оператор `NewBitmap->LoadFromFile( FileName );` — оператором `NewBitmap->LoadFromResourceID( HInstance, ResourceID );`.

Следующий этап — создание нового интерфейса IDirectDrawSurface3 для хранения растрового изображения. Программная реализация этого процесса показана в листинге 26.20.

Листинг 26.20. Создание DirectDraw-поверхности на основе компонента TBitmap

```
bool CreateSurfaceFromTBitmap( LPDIRECTDRAWSURFACE3 *pSurface,
                               Graphics::TBitmap *Bitmap )
{
    // Этот компонент будет указывать на новую поверхность.
    TCanvas *SurfaceCanvas;
    // Здесь будет содержаться описание новой поверхности.
    DDSURFACEDESC SurfaceDescription;
    HRESULT DDError;
    HDC hDeviceContext;

    // Обнуляем память, содержащую описание поверхности.
    ZeroMemory(&SurfaceDescription, sizeof( DDSURFACEDESC ));

    // Устанавливаем член, "отвечающий" за размер описания.
    SurfaceDescription.dwSize = sizeof( DDSURFACEDESC );

    // Устанавливаем флаги для индикации того, что поля
    // характеристик, ширины и высоты действительны
    SurfaceDescription.dwFlags =
                          DDSD_CAPS | DDSD_WIDTH | DDSD_HEIGHT;

    // Устанавливаем свойство dwCaps для создания плоской
    // поверхности.
    SurfaceDescription.ddsCaps.dwCaps = DDSCAPS_OFFSCREENPLAIN;

    // Устанавливаем размеры поверхности в соответствии с шириной
    // и высотой компонента Bitmap.
    SurfaceDescription.dwWidth = Bitmap->Width;
    SurfaceDescription.dwHeight = Bitmap->Height;

    // Делаем попытку создать поверхность.
    DDError = DirectDraw2Interface->CreateSurface(
                       &SurfaceDescription,
                       (LPDIRECTDRAWSURFACE *) pSurface, NULL );

    // В случае неудачи...
    if(FAILED( DDError ))
    {
        // ...возвращаем признак ошибки.
        return( false );
    }

    // В противном случае копируем растровое изображение  Bitmap
    // на поверхность.
    // Для этого получаем контекст устройства для поверхности.
    DDError = (*pSurface)->GetDC( &hDeviceContext );
```

```
     // В случае неудачи...
    if(FAILED( DDError ))
    {
        // ...возвращаем признак ошибки.
        return( false );
    }

    // Создаем новую канву для доступа к поверхности.
    SurfaceCanvas = new TCanvas();

    // Связываем новую канву с контекстом устройства поверхности.
    SurfaceCanvas->Handle = hDeviceContext;

    // Накладываем растровое изображение на канву поверхности.
    SurfaceCanvas->Draw( 0, 0, Bitmap );

    // Освобождаем контекст устройства поверхности.
    (*pSurface)->ReleaseDC( hDeviceContext );

    // Освобождаем компонент TCanvas.
    delete SurfaceCanvas;

    // Возвращаем признак успешного выполнения.
    return( true );
}
```

На заметку Безусловно, параметр pSurface при передаче этой функции должен быть свободным (незанятым).

Теперь у нас есть три поверхности: главная (первичная) поверхность дисплея, которую видит пользователь, поверхность заднего буфера, на которой выполняется рисование, и растровая поверхность (причем растровых поверхностей может быть несколько), содержащая спрайт (элемент динамического графического изображения), который мы желаем отобразить. Осталось лишь увязать все это. В листинге 26.21 показано, как 1000 раз вывести растровое изображение в задний буфер.

Листинг 26.21. Демонстрационная процедура использования растровых изображений

```
// При вызове процедуры BitmapDemo следующие переменные должны
// быть инициализированы глобально или в классе.
LPDIRECTDRAWSURFACE3 pBitmapSurface;
LPDIRECTDRAWSURFACE3 pPrimarySurface, pBackBuffer;


bool TForm1::BitmapDemo( void )
{
    // Счетчик цикла.
    int i;
    // Координаты x и y для рисования.
    int x, y;
```

```
    // Максимально возможные координаты x и y, позволяющие
    // растровому изображению оставаться в заднем буфере.
    int maxX, maxY;
    // Структуры для хранения информации об используемых поверхностях.
    // DDSURFACEDESC BackBufferDesc, BitmapDesc;
    // Переменная для хранения значения ошибки.
    HRESULT DDError;

    // Как всегда, обнуляем память.
    ZeroMemory(&BackBufferDesc, sizeof(DDSURFACEDESC));
    // Как всегда, устанавливаем член, "отвечающий" за размер
    // поверхности заднего буфера.
    BackBufferDesc.dwSize=sizeof(DDSURFACEDESC);
    // Получаем описание поверхности, запоминаемое в заднем
    // буфере (мы хотим знать его ширину и высоту).

    DDError = pBackBuffer->GetSurfaceDesc(&BackBufferDesc);

    // В случае неудачи возвращаем признак ошибки.
    if(FAILED( DDError ))
        return( false );

    // Мы также хотим знать ширину и высоту растровой поверхности:
    // ZeroMemory(&BitmapDesc, sizeof(DDSURFACEDESC));
    BitmapDesc.dwSize=sizeof(DDSURFACEDESC);
    DDError = pBitmapSurface->GetSurfaceDesc(&BitmapDesc);

    // В случае неудачи возвращаем признак ошибки.
    if(FAILED( DDError ))
        return( false );

    // Вычисляем максимально возможные координаты x и y
    // при сохранении всей растровой поверхности внутри заднего буфера.
    maxX=BackBufferDesc.dwWidth-BitmapDesc.dwWidth;
    maxY=BackBufferDesc.dwHeight-BitmapDesc.dwHeight;

    // Выполняем цикл 1000 раз.
    for(i=0; i<1000; i++)
    {
        // Получаем произвольные координаты.
        x=random( maxX );
        y=random( maxY );
        // Выводим растровое изображение в задний буфер.
        pBackBuffer->BltFast(x, y, pBitmapSurface, NULL, DDBLTFAST_NOCOLORKEY);
    }
    return( true );
}
```

Если просмотреть содержимое листинга 26.21, нетрудно заметить вызов функции-члена `BltFast()` интерфейса `IDirectDrawSurface2` (список параметров функции-члена `BltFast()`

приведен в табл. 26.3). Эта функция по сути копирует прямоугольную область пикселей с одной поверхности в другую, причем это делается с поразительной скоростью и гибкостью.

```
HRESULT IDirectDrawSurface2::BltFast( DWORD x, DWORD y,
                                      LPDIRECTDRAWSURFACE3 SourceSurface,
                                      LPRECT SourceRect, DWORD Flags );
```

Таблица 26.3. Параметры функции `BltFast()`

Имя параметра	Описание
`x`	Координата x приемника
`y`	Координата y приемника
`SourceSurface`	Указатель на поверхность `DirectDrawSurface`, с которой выполняется копирование
`SourceRect`	Прямоугольная область копируемых пикселей (значение `NULL` используется для всей поверхности)

`Flags`	Параметр `Flags` функции `BltFast()` позволяет задавать способ копирования растра. Возможными значениями параметра `Flags` являются следующие.
	`DDBLTFAST_DESTCOLORKEY`. Используется цветовой тон принимающей поверхности `DirectDrawSurface`
	`DDBLTFAST_SRCCOLORKEY`. Используется цветовой тон исходной поверхности `DirectDrawSurface`
	`DDBLTFAST_NOCOLORKEY`. Цветовой тон не используется
	`DDBLTFAST_WAIT`. Функция возвратит управление только по завершении переноса растрового изображения или при возникновении ошибки

Цветовой тон (в данном контексте) означает цвет, который не копируется. Он используется, чтобы спрайт (который не является прямоугольником) можно было переносить со специфичным цветом (если таковой присутствует), который "отвечает" за прозрачность (например, черный), и в этом случае цветовой тон устанавливается равным этому цвету. Можно также для цветового тона установить диапазон значений (например, от черного до синего, как это сделано в следующем примере). Установка цветового тона для поверхности показана в листинге 26.22.

Листинг 26.22. Установка цветового тона для поверхности

```
// Содержит информацию о цветовом тоне.
DDCOLORKEY ColorKey;

// Обнуляем память.
ZeroMemory( &ColorKey, sizeof( DDCOLORKEY ) );
// Устанавливаем самое низкое значение для цветового тона (темно-синий цвет).
ColorKey.dwColorSpaceLowValue = RGB(0, 0, 0);
ColorKey.dwColorSpaceHighValue = RGB(0, 0, 255);
// Устанавливаем цветовой тон для поверхности.
DDError = Surface->SetColorKey(DDCKEY_COLORSPACE | DDCKEY_SRCBLT, &ColorKey);
// DDCKEY_COLORSPACE уведомляет SetColorKey, что мы устанавливаем
// цветовой диапазон.
// DDCKEY_SRCBLT уведомляет SetColorKey , что мы устанавливаем
// цветовой тон исходной поверхности.
```

Пример DirectDraw-программы

Пример DirectDraw-программы (файл проекта называется `DirectDraw.bpr`) находится в папке `DirectDraw` на компакт-диске, прилагаемом к этой книге. Несмотря на простоту, программа демонстрирует все основные методы, представленные в этом разделе.

Сначала инициализируется интерфейс DirectDraw и устанавливается соответствующий кооперативный уровень и разрешение экрана. Затем на DirectDraw-поверхность загружается растровое изображение, которое выводится в главный буфер (Primary Buffer) 1000 раз. После отображения текста поверхности меняются местами. Для выхода из программы пользователь может нажать любую клавишу.

Если вы выберете команду Project⇨Options (Проект⇨Параметры), а затем откроете вкладку Directories/Conditionals (Каталоги/Условия), то заметите, что к пути библиотеки добавлено выражение пути `$(BCB)\lib\PSDK`. В этом проекте используется библиотека `ddraw.lib`, предоставляемая C++Builder, и в этом можно убедиться, заглянув в исходный файл `DirectDraw.cpp` и в файл проекта `DirectDraw.bpr`. Эта библиотека просто позволяет избежать задания абсолютного пути.

Для своих DirectDraw-приложений вы можете просто выбрать из главного меню команду Project⇨Add to Project (Проект⇨Добавить в проект), затем в комбинированном списке Files of Type (Тип файла) отыскать вариант Library File (*.lib) и найти файл `ddraw.lib` в папке `\Lib\PSDK`, где был инсталлирован C++Builder.

Резюме по DirectDraw

Не так уж и мало для введения, не так ли? Вам остается лишь воплотить рассмотренные здесь методы во что-нибудь полезное.

Обработчик событий `OnIdle()` можно использовать в приложении для обновления кадров, как описано в разделе “Циклы рисования в среде C++Builder, реализуемые с помощью функции `OnIdle()`” этой главы. Описанных здесь процедур вполне достаточно, чтобы вы могли реализовать игры с быстрой сменой декораций. Если вас интересует работа DirectDraw в окне, то используемые в этом случае методы будут несколько иными, но не сложнее уже рассмотренных.

Использование DirectSound

DirectSound — аудиоэквивалент DirectDraw. Это прямой путь к аудиоаппаратуре в системе с инсталлированным интерфейсом DirectX. Он работает очень быстро и имеет малое время ожидания (события, связанные со звуком, происходят после определенного сигнала, а не вследствие системной “подачи”).

Между DirectDraw и DirectSound есть много общего.

- Оба интерфейса базируются на COM-архитектуре.
- Каждый из них имеет главный интерфейс и субинтерфейс: IDirectDraw и IDirectDrawSurface, соответственно, для DirectDraw; IDirectSound и IDirectSoundBuffer, соответственно, для DirectSound.
- Оба могут иметь кооперативный уровень, установленный для достижения различных уровней управления аппаратурой.

Однако закончим разговоры и примемся за инициализацию интерфейса DirectSound.

Инициализация объекта DirectSound

Чтобы создать объект DirectSound, мы используем вспомогательную функцию `DirectSoundCreate()`. Этой функции нужно передать указатель на объект интерфейса и GUID нужного устройства. Если вас устроит стандартное устройство, то, как и в случае с функцией `DirectDrawCreate()`, можно в качестве GUID передать значение `NULL`. После создания интерфейса можно установить кооперативный уровень. Обычно он устанавливается равным значению `DSSCL_NORMAL`, если вы создаете приложение для работы в окне. Это позволит другим программам для воспроизведения звука использовать то же самое устройство. Если вы пишете полноэкранное приложение или игру, вам стоит использовать значение `DSSCL_EXCLUSIVE`, чтобы другие приложения в это время безмолвствовали.

В листинге 26.23 приведена короткая программа, иллюстрирующая создание объекта `IDirectSound` и установку кооперативного уровня.

Листинг 26.23. Инициализация объекта DirectSound

```
// Объявляется глобально или в классе.
LPDIRECTSOUND DirectSoundInterface;

// Необходимо передать дескриптор (типа HWND) главного окна приложения.
// Примечание:
//    Если используется также и интерфейс DirectDraw, нужно
//    передать тот же самый дескриптор окна.
//
bool InitializeDirectSound( HWND WindowHandle )
{
    // Эта переменная будет содержать любой ошибочный результат.
    HRESULT DError;

    // Попробуем захватить стандартное звуковое устройство.
    DError = DirectSoundCreate( NULL, &DirectSoundInterface, NULL );

    // В случае неудачи...
    if( FAILED(DError) )
    {
        // ...возвращаем признак ошибки.
        return( false );
    }

    // Попробуем установить нормальный кооперативный уровень.
    DError = DirectSoundInterface->SetCooperativeLevel(WindowHandle,
                                                       DSSCL_NORMAL);
    // В случае неудачи...
    if( FAILED(DError) )
    {
        // ...освобождаем объект IDirectSound.
        DirectSoundInterface->Release();
        // Возвращаем признак ошибки.
        return( false );
    }
```

```
    // В противном случае возвращаем признак успешного выполнения.
        return( true );
}
```

С объектом `IDirectSound` связан еще один объект, который называется главным (первичным) буфером и автоматически создается и поддерживается интерфейсом DirectX. Его назначение — вывод звуковых (wave) данных. Программист сообщает DirectSound, что поместить в этот буфер и как микшировать. Объекты, называемые вторичными буферами, используются для хранения звуковых выборок, и при воспроизведении звука отправляют их в первичный буфер DirectSound. После создания объекта `IDirectSound` у вас есть возможность изменить формат звуковых данных первичного буфера (частоту дискретизации, количество каналов и число битов на выборку). К сожалению, ограничения книжной площади не позволяют углубиться в описание этого процесса (хотя в нем нет ничего сложного). За подробностями обращайтесь к документации по DirectX.

Создание вторичного буфера

Процесс создания вторичного звукового буфера состоит из трех этапов.

- Загрузка звуковых данных в плоский массив. Эти данные можно загрузить из файла или ресурсной составляющей программы.
- Создание звукового буфера с использованием объекта `IDirectSound`.
- Анализ звуковых данных в звуковом буфере.

Каждому этапу мы уделим должное внимание и продемонстрируем на примере полный процесс.

Этап 1: загрузка звуковых данных

Рассмотрим загрузку звуковых данных из WAV-файла. Для этого достаточно скопировать содержимое этого файла в массив, который затем используется на третьем этапе (при анализе звуковых данных в звуковом буфере). Фрагмент программы, при выполнении которого происходит загрузка файла, представлен в листинге 26.24.

Листинг 26.24. Загрузка WAV-файла в память

```
bool LoadWAVFromFile( char *&Data, int &Size, char *FileName )
{
    // Указатель на файл.
    FILE *fp;
    // Эта переменная будет содержать длину файла.
    fpos_t EndOfFilePos;

    // Открываем файл для чтения в двоичном режиме.
    fp = fopen(FileName, "rb");
    // В случае неудачи возвращаем признак ошибки.
    if( fp == NULL )
    {
        return( false );
    }
    // Находим конец файла.
    fseek(fp, 0, SEEK_END);
```

```
    // Получаем позицию в конце файла (т.е. длину файла).
    fgetpos(fp, &EndOfFilePos);
    // Снова находим начало файла.
    fseek(fp, 0, SEEK_SET);

    // Устанавливаем размер.
    Size=(int)EndOfFilePos;

    // Выделяем буфер для размера файла.
    Data = new char[(long)EndOfFilePos];
    // Если не хватило памяти...
    if( Data == NULL )
    {
        // ...закрываем файл и...
        fclose(fp);
        // ... возвращаем признак ошибки.
        return( false );
    }
    // Читаем данные.
    if( (signed)fread( Data, 1, EndOfFilePos, fp ) != EndOfFilePos )
    {
        // Если мы не прочитали верный объем данных, то закрываем файл,...
        fclose(fp);
        // ...освобождаем буфер...
        delete []Data;
        // ...и возвращаем признак ошибки.
        return( false );
    }
    // Закрываем файл и возвращаем признак успешного выполнения.
    fclose(fp);
    return( true );
}
```

Теперь у нас есть плоский char-массив, который содержит все данные из WAV-файла, включая заголовок и реальную звуковую информацию.

Этап 2: создание вторичного буфера

Этот этап довольно прост. Объект IDirectSound используется для создания указателя на интерфейс IDirectSoundBuffer, что и показано в листинге 26.25.

Листинг 26.25. Создание вторичного буфера DirectSound

```
// Объявляется глобально или в классе:
LPDIRECTSOUND DirectSoundInterface;

bool CreateSecondaryBuffer( LPDIRECTSOUNDBUFFER *DirectSoundBuffer,
                            DSBUFFERDESC &DSBufferDescription )
{
    // Эта переменная предназначена для хранения признака ошибки.
    HRESULT DSError;
```

```
    // Устанавливаем член размера.
    DSBufferDescription.dwSize = sizeof(DSBUFFERDESC);
    // Устанавливаем член флагов.
    DSBufferDescription.dwFlags = DSBCAPS_STATIC | DSBCAPS_GLOBALFOCUS;
    // Создаем звуковой буфер.
    DSError = DirectSoundInterface->CreateSoundBuffer(
                                &DSBufferDescription, DirectSoundBuffer, NULL);
    // В случае неудачи возвращаем признак ошибки.
    if( FAILED( DSError ) )
    {
        return( false );
    }
    return ( true );
}
```

Нетрудно заметить, что член `dwFlags` описания `DSBUFFERDESC` устанавливается равным значению `DSBCAPS_STATIC | DSBCAPS_GLOBALFOCUS`. Первый флаг сообщает интерфейсу DirectSound, что буфер должен быть статически загружен в память. Второй говорит о том, что звук может воспроизводиться, когда само приложение работает в фоновом режиме.

Этап 3: анализ звуковых данных в прямом буфере

Это самый сложный этап, на котором программа будет анализировать стандартный WAV-файл. Сначала необходимо убедиться (программа это и делает), что файл (который на самом деле представляет собой буфер в памяти) имеет корректный формат. Затем программа читает содержимое буфера и извлекает информацию о формате и звуковые данные. Все это показано в листинге 26.26.

Листинг 26.26. Извлечение заголовка и данных из звукового буфера

```
bool ParseWavBuffer( char *&pBuffer, WAVEFORMATEX **ppWaveHeader,
               unsigned char **ppbWaveData, DWORD *&pdwWaveSize )
{
    // Эта переменная - DWORD-указатель на текущую позицию в буфере.
    DWORD *pdw;
    // Переменная pdwEnd - это DWORD-указатель на конец буфера.
    DWORD *pdwEnd;
    // Переменная dwRiff содержит некоторый специальный RIFF-код.
    DWORD dwRiff;
    // Переменная dwType содержит тип файла для этого RIFF-кода.
    DWORD dwType;
    // Переменная dwLength содержит длину данных.
    DWORD dwLength;

    if(pBuffer==NULL)
        return(false);
    // Убеждаемся в том, что указатели установлены равными NULL.
    if(ppWaveHeader!=NULL)
        *ppWaveHeader=NULL;
    if(ppbWaveData!=NULL)
        *ppbWaveData=NULL;
    if(pdwWaveSize)
```

```
    *pdwWaveSize=0;

// Устанавливаем DWORD-указатель на буфер.
pdw=(DWORD *)pBuffer;
// Получаем содержимое формата RIFF-файла.
dwRiff=*pdw++;
// Получаем длину данных.
dwLength=*pdw++;
// Получаем ID типа RIFF-файла.
dwType=*pdw++;

// Проверка того, что это действительно RIFF-файл.
if(dwRiff!=mmioFOURCC('R', 'I', 'F', 'F'))
{
    return(false);
}
// Проверка того, что это действительно WAVE-файл.
if(dwType!=mmioFOURCC('W', 'A', 'V', 'E'))
{
    return(false);
}
// Получаем указатель на конец данных.
pdwEnd=(DWORD *)((unsigned char *)pdw+dwLength-4);

// Работаем в цикле, пока не достигнем конца данных.
while(pdw<pdwEnd)
{
    // Получаем тип раздела.
    dwType=*pdw++;
    // Получаем длину раздела.
    dwLength=*pdw++;

    switch(dwType)
    {
    // Если этот раздел содержит информацию о формате...
    case mmioFOURCC('f', 'm', 't', ' '):
        // Если мы все-таки не получили информацию о формате...
        if(ppWaveHeader && !*ppWaveHeader)
        {
            // Если длина этого формата не такая, как должна быть...
            if(dwLength<sizeof(WAVEFORMAT))
            {
                // ... возвращаем признак ошибки.
                return(false);
            }
            // Устанавливаем Wave-заголовок, чтобы указать,
            // что это часть данных.
            *ppWaveHeader=(WAVEFORMATEX *)pdw;
            // Если у нас есть данные и заголовок, возвращаем признак true.
            if((!ppbWaveData||*ppbWaveData) &&
                (!pdwWaveSize ||* pdwWaveSize))
```

```
                return(true);
            }
            break;
        // Если этот раздел содержит данные...
        case mmioFOURCC('d', 'a', 't', 'a'):
            // Если мы все-таки не получили данных...
            if((ppbWaveData && !*ppbWaveData) ||
               (pdwWaveSize && !* pdwWaveSize))
            {
                // Получаем указатель на данные.
                if(ppbWaveData)
                    *ppbWaveData=(unsigned char *)pdw;
                // Получаем длину данных.
                if(pdwWaveSize)
                    * pdwWaveSize =dwLength;
                // Если мы получили все, что нужно, возвращаемся.
                if(!ppWaveHeader || *ppWaveHeader)
                    return(true);
            }
            break;
        }
        // Переходим к следующей границе слова.
        pdw=(DWORD *)((unsigned char *)pdw+((dwLength+1)&~1));
    }
    // Если мы попали сюда (чего не должно было случиться),
    // возвращаем признак ошибки.
    return(false);
}
```

А теперь объединим функции WAV-загрузчика

Теперь нам нужна функция, которая бы свела описанные выше действия в единое целое. Итак, сначала нам нужно загрузить WAV-файл в два буфера: один для WAV-данных, а другой — для заголовка. Затем необходимо создать объект интерфейса `IDirectSoundBuffer`, заблокировать его, загрузить в него данные, после чего разблокировать. Все это реализовано в программе, представленной в листинге 26.27.

Листинг 26.27. Загрузка WAV-файла в буфер интерфейса DirectSound

```
bool BCBDirectSound::LoadWAVIntoDirectSoundBuffer(
                     LPDIRECTSOUNDBUFFER *DSBuffer, char *FileName )
{
    // Указатель на загруженный WAV-файл.
    char *WAVFile;
    // Размер загруженного WAV-файла.
    int WAVFileSize;
    // Новое описание DirectSoundBuffer.
    DSBUFFERDESC NewSoundBufferDescription;
    // Реальные звуковые данные, выделенные из файла.
    unsigned char *WaveData;
    // Указатели на заблокированный буфер DirectSoundBuffer.
```

```
LPVOID pMem1, pMem2;
// Размеры заблокированных указателей.
DWORD dwSize1, dwSize2;
// Переменная для хранения признака ошибки.
HRESULT DSError;

// Инициализируем WAVFile.
WAVFile=NULL;

// Обнуляем описание звукового буфера.
ZeroMemory(&NewSoundBufferDescription, sizeof(DSBUFFERDESC));

// Если мы не можем загрузить wav-файл, возвращаем признак ошибки.
if(!LoadWAVFromFile( WAVFile, WAVFileSize, FileName ))
    return( false );
// Если мы не можем разбить wav-файл на составляющие,
// возвращаем признак ошибки.
if(!ParseWavBuffer( WAVFile, &NewSoundBufferDescription.lpwfxFormat,
                  &WaveData, &NewSoundBufferDescription.dwBufferBytes ))
{
    delete WAVFile;
    return( false );
}
// Если мы не можем создать объект DirectSoundBuffer,
// возвращаем признак ошибки.
if(!CreateSecondaryBuffer( DSBuffer, NewSoundBufferDescription ))
{
    delete WAVFile;
    return( false );
}
// Попробуем заблокировать память DirectSoundBuffers.
DSError = (*DSBuffer)->Lock(0, NewSoundBufferDescription.dwBufferBytes,
                            &pMem1, &dwSize1, &pMem2, &dwSize2, 0);
// В случае неудачи возвращаем признак ошибки.
if(FAILED(DSError))
{
    delete WAVFile;
    return( false );
}
// Копируем первый фрагмент заблокированной памяти.
memcpy(pMem1, WaveData, dwSize1);
// Если второй фрагмент пришлось заблокировать, выполняем копирование.
if(dwSize2!=0)
    memcpy(pMem2, WaveData+dwSize1, dwSize2);
// Разблокируем память.
(*DSBuffer)->Unlock(pMem1, dwSize1, pMem2, dwSize2);

// Освобождаем память.
delete WAVFile;
```

```
    // Возвращаем признак отсутствия ошибки.
    return ( true );
}
```

Теперь у нас есть в полной мере функциональный интерфейс IDirectSoundBuffer. В остальном — все чрезвычайно просто. Чтобы "озвучить" буфер, достаточно выполнить следующее.

```
DSError = DSBuffer->Play(res1, res2, how);
```

Первые два параметра зарезервированы и должны быть равны 0 (нулю). Последний параметр определяет, как именно вы бы хотели воспроизвести содержимое буфера. Можно задать число 0, означающее однократное воспроизведение с последующим остановом, или значение DSBPLAY_LOOPING для проигрывания в бесконечном цикле. Возвращаемое значение имеет тип HRESULT. С помощью макроса FAILED() можно определить, насколько успешной была команда Play().

Описанный выше метод имеет одну проблему, которая заключается в невозможности еще одного воспроизведения "поверх" уже воспроизводимой версии. В этом случае просто продолжается начатое воспроизведение. Это вполне подходит для звуков, которые могут звучать только "соло" (например, речь или щелчок на кнопке). Однако у вас все же может возникнуть надобность в проигрывании некоторых звуков поверх их же копий. В интерфейсе IDirectSound предусмотрена функция копирования объектов IDirectSoundBuffers, которая позволяет решить эту задачу.

Пример DirectSound-программы — „многоголосый" плейер

В папке DirectSound на прилагаемом к книге компакт-диске, вы найдете пример программы, где демонстрируются все методы DirectSound, описанные в этой главе. Программа реализации звуковоспроизведения включена в модуль BCBDirectSound.

Сначала программа инициализирует интерфейс DirectSound. Затем обработчик событий OnIdle() приложения настраивается на вызов функции UpdatePlayingSounds(). Эта функция просматривает множество воспроизводимых звуков и удаляет те, которые уже завершились. Пользователь может загружать WAV-файлы с помощью меню File, а для их воспроизведения достаточно щелкнуть на нужном имени.

Аналогично DirectDraw-программе, в путь библиотеки для данного проекта был добавлен элемент $(BCB)\lib\PSDK. В этом проекте используется файл библиотеки dsound.lib, включенный в комплект C++Builder.

Для своих DirectSound-приложений вы можете просто выбрать из главного меню команду Project⇨Add to Project (Проект⇨Добавить в проект), затем в комбинированном списке Files of Type (Тип файла) отыскать вариант Library File (*.lib) и найти файл dsound.lib в папке \Lib\PSDK, в которой был инсталлирован C++Builder.

Еще несколько слов о DirectX

Помимо интерфейсов DirectDraw и DirectSound, которые считаются самыми простыми в работе API, DirectX включает другие интерфейсы для работы с множеством устройств и мультимедийных форматов. Следующий список составлен из других API, входящих в состав DirectX.

- DirectInput — это интерфейс DirectX API, предназначенный для обслуживания процесса ввода данных с малым временем ожидания (с помощью клавиатуры, мыши или джойстика).

- DirectPlay — сетевой интерфейс API. Он разработан для упрощения организации связи посредством Internet, LAN (local area network — локальная сеть) или любого последовательного соединения.
- Direct3D — это DirectX-ответ компании Microsoft на создание высокоскоростного инструмента (с аппаратной поддержкой) для построения трехмерных графических изображений в режиме реального времени. Этот интерфейс разработан для программирования компьютерных игр.

Дополнительные источники информации по DirectX

Это большая тема, и вам вряд ли удастся ограничиться одним источником информации. Скорее всего жизнь (рано или поздно) заставит вас обратиться к дополнительной литературе по программированию DirectX. Помимо книг, посвященных этой теме, существует немало других источников информации, подготовленных компанией Microsoft. Эти источники мы и предлагаем вашему вниманию.

- *Inside DirectX*, Bradley Bargen, Peter Donnelly, ISBN 1-57231-696-9, Microsoft Press.

  Это самое полное руководство по DirectDraw, DirectSound, DirectPlay и DirectInput.
- Справочные файлы MSDN SDK.

  Они распространяются при любой подписке на MSDN и освещают все аспекты работы с DirectX.
- Список почтовой рассылки разработчиков DirectX (адрес DirectX Developers: `DIRECTXDEV@discuss.microsoft.com`).

  Здесь вы узнаете мнение профессионалов — основных разработчиков DirectX и членов команд, усилиями которых был создан DirectX.
- Web-узел центра разработчиков DirectX (MSDN DirectX Developer Center) по адресу: `http://msdn.microsoft.com/DirectX`.

  Это основной Internet-ресурс для разработчиков DirectX.

Резюме

В этой главе были рассмотрены основные возможности интерфейсов OpenGL, DirectDraw и DirectSound API. Этой теме посвящены целые книги, поэтому, если вам нужна дополнительная информация, вы всегда можете найти ее в этих книгах, а также на упомянутых выше Web-узлах в списках рассылки.

Используя полученные знания, вы сможете объединить в любом приложении Borland C++Builder 5 высококачественную графику и звук. Однако важно помнить, что компьютер, на котором предположительно будет работать разрабатываемая вами DirectX-программа, должен иметь инсталлированные драйверы соответствующей версии, в противном случае эти команды не будут работать. Всегда проверяйте значения, возвращаемые командами DirectX, которые могут завершиться неудачей (инициализация, создание интерфейса и пр.). И хотя функции OpenGL при возникновении ошибок не вызовут аварийных ситуаций, но последствия такой работы предугадать трудно.

Развертывание C++Builder-приложений

ЧАСТЬ

V

СОЗДАНИЕ СПРАВОЧНЫХ ФАЙЛОВ И ДОКУМЕНТАЦИИ
РАСПРОСТРАНЕНИЕ ПРОГРАММНЫХ ПРОДУКТОВ
ИНСТАЛЛЯЦИЯ И ОБНОВЛЕНИЕ ПРОГРАММНЫХ ПРОДУКТОВ

Глава

27

Создание справочных файлов и документации

Дру Эйвис

Вам, конечно же, по собственному опыту известно, каково это: разыскивать какую-нибудь информацию сначала с помощью интерактивной справки, а затем — в документации. Возможно, вы пытались найти необходимую для завершения задачи процедуру или выяснить причину сообщения об ошибке. Искали — и не нашли (по причине ее отсутствия) или, что еще хуже, нашли, но информация оказалась явно неверной. Предоставляемая вами документация — единственный посредник между приложением и его пользователями, который может облегчить освоение ими программного продукта. Хорошая документация является важным фактором, который учитывает пользователь, принимая решение о покупке программного обеспечения. Кроме того, именно документация служит основой для создания "образа" вашего приложения при подготовке обзора программных продуктов в средствах массовой информации.

Прекрасно иллюстрирует важность документации случай, произошедший в компании Microsoft во время работы над Word 97. Для того чтобы выяснить, какие новые функции следует внести в будущую версию, были опрошены пользователи предыдущей версии (Word 95), и оказалось, что 80% упомянутых функций уже были реализованы в старом продукте. Из этого следуют два вывода: полезные средства могут быть глубоко погребены в слабеньком пользовательском интерфейсе, а плохая справочная документация упрятывает их еще глубже.

Любое приложение, независимо от его размеров, должно сопровождаться справочной документацией. В данной главе будут описаны различные ее форматы, которые вы сможете предложить пользователям, а также некоторые принципы составления документации.

На заметку

Если вы занимаетесь разработкой справочных документов для большой организации, вполне возможно, что вам не стоит тратить время на эту главу, и вы можете ее пропустить. Можно считать, что в этом смысле вам повезло: наверняка у вашей компании есть возможность обратиться к профессиональным авторам технической литературы, людям, съевшим собаку на составлении такой документации. Если у вас все же нет такого доступа (т.е. если вы работаете на себя, отвечая за весь проект, или в небольшой фирме), прислушайтесь к нашему совету: оцените возможность найма специалиста для выполнения этой части работы. Множество организаций, например Общество по созданию технической информации (Society for Technical Communication) или специальная группа по созданию технической информации Института инженеров по электротехнике и электронике (IEEE (Institute of Electrical and Electronics Engineers) Technical Communication SIG (special interest group)), имеют филиалы в разных городах и могут предложить вам целый список таких специалистов.

С другой стороны, если вы "сам себе режиссер и исполнитель", т.е. специалист по маркетингу, агент по продажам и обслуживанию клиентов и одновременно директор по финансовым вопросам, усядьтесь поудобнее и настройтесь на внимательное чтение следующих разделов, так как вам предстоит стать еще и составителем технической документации.

Создание технической документации, или десять верных шагов на пути к совершенству

Прежде чем с головой зарыться в изучение всех премудростей создания справочных систем и руководств пользователя, постарайтесь понять, что именно сделает техническую документацию хорошей.

- *Изучите свою аудиторию*. Это самый первый принцип создания хорошей документации. Кто ваши пользователи? Что они надеются найти, обращаясь к вашей документации?
- *Запомните три основных условия создания технической документации*. Сделайте ее краткой, полной и достоверной.

- *Придерживайтесь определенного стиля*. В кругах распространения технической информации слово “стиль” употребляется как синоним слова “согласованность”. Станете ли вы, к примеру, выделять важные моменты в одной главе жирным шрифтом, а в другой курсивом? Выработайте единые стилевые правила и следуйте им во всех разделах документации. Microsoft Press издает специальное руководство по стилевому оформлению документации на программные продукты. Чикагское издание (*The Chicago Manual of Style*) — прекрасное руководство по стилю общего назначения.
- *С самого начала отведите время для документации*. Помните, что для написания документации требуется немалое время, поэтому запланируйте этот этап в качестве составной части процесса разработки.
- *Критично относитесь к собственной писанине, а лучше найдите кого-нибудь, кто раскритикует ее еще больше*. Желательно, чтобы этот “кто-то” не провалил экзамен по родному языку в восьмом классе средней школы.
- *Одна картинка может заменить несколько сотен слов*. Используйте графические изображения, если они предоставляют дополнительную информацию, которая отсутствует в тексте или же существенно сокращают его объем. Если же вы вставили картинку лишь потому, что она мило выглядит, скорей от нее избавьтесь. Если она, кроме того, занимает 500 Кбайт, избавьтесь от нее еще скорей!
- *Хотя бы раз просмотрите все разделы с описанием последовательности действий*. Убедитесь, что вы не пропустили какое-нибудь важное действие или не добавили ненужное.
- *Всегда предупреждайте пользователя о том, что должно произойти в результате выполнения очередного действия*. Не ограничивайтесь, например, простым указанием щелкнуть на кнопке OK; сообщите, что случится после этого.
- *Обязательно предупредите пользователя об опасности потери данных или повреждения системы*. Эта информация непременно должна предшествовать описанию опасной по своим последствиям инструкции или рискованной процедуры.
- *Отнеситесь к документации как к одной из важнейших составляющих своего приложения*. Это единственная связующая нить между вашим пользователем и интерфейсом.

Типы документации

Помните, как в старые добрые времена программное обеспечение сопровождалось одним единственным видом документации — руководством пользователя. Такой, часто плохо организованный справочник, представлял собой жалкую хаотичную смесь из инструкций к различным командам и ссылок на ошибки (никак не упорядоченные). В редких случаях приводились наиболее вероятные ошибки пользователя. Но времена изменились, компьютерная индустрия далеко продвинулись, а вместе с ней на более высокий уровень поднялась и техническая документация. Сегодня в нашем распоряжении целый набор инструментальных средств технической поддержки, сопровождающих программное обеспечение. Ниже приводится краткий список распространенных типов документации, разделенный по признаку получения информации (на бумаге или в электронном варианте).

В печатной форме:

- Руководство пользователя
- Руководство по устранению неисправностей

- Справочник
- Руководство для начинающих
- Список клавиатурных эквивалентов команд
- Краткий справочник

В электронном виде:

- Справочная система
- Контекстно-зависимая справка
- Мастера
- Учебное пособие (самоучитель)
- Ежедневные советы
- Экранные подсказки
- Что это такое?
- Файл Readme (Прочти меня)/Сведения о программе
- Web-узел
- База знаний

По мере того как печать дорожает, а сила и мощь интерактивных справочных систем растет, создатели программного обеспечения все реже прибегают к традиционным бумажным руководствам и справочникам. Вместо этого разработчики составляют большую часть документации в электронном виде, а печатные руководства уходят на задний план с тенденцией к полному исчезновению. Фирма Microsoft поддалась этой тенденции, выпустив Windows 95 без всякой печатной документации, если не считать тоненькое руководство по инсталляции. Но, к сожалению, интерактивная справочная документация оказалась в данном случае не очень удачной, поэтому вряд ли стоит рассматривать этот случай как пример для подражания.

Стратегии создания интерактивной документации

Перед началом составления документации желательно наметить план с описанием предполагаемых типов справки, которые могут понадобиться пользователю, и способы их реализации. Приступая к этой работе, задайте себе вопрос*: Кто станет использовать это приложение?* Проанализируйте потребительский рынок. Но не забывайте, что многие пользователи для решения большинства задач будут искать в вашей документации описание последовательности действий (или процедурную информацию — см. раздел “Процедурная справка” ниже в этой главе).

Затем подумайте, какой вид справочной и понятийной информации понадобится пользователям? Что лучше: скомпоновать разделы в основном справочном файле (например, упорядочив их по алфавиту, как в словаре) или организовать их в виде отдельных файлов (как синтаксические ссылки)? Как связать процедурную информацию со справочной и наоборот? И, наконец, решите, стоит ли включать в разделы инструкций или учебные главы примеры, которые должны помочь пользователям освоить ваше приложение.

И хотя принципы составления хорошей документации в печатном виде можно перенести на электронную справку, между ними существуют определенные отличия, которые проявляются прежде всего в самой организации электронной справки.

Категории справочной информации

Всю справочную информацию можно условно разделить на несколько основных категорий, каждая из которых будет удовлетворять различные потребности пользователя в справке:

- процедурная
- справочная
- концептуальная;
- учебная.

Процедурная информация

Возможно, из-за того, что процедурная справка (описание последовательности действий) помогает быстро решить конкретную задачу и отвечает всеобщей потребности быстро добиться желаемого, она является наиболее распространенной из всех видов справки, предоставляемых пользователю (рис. 27.1). Хорошая процедурная справка начинается с известной пользователю "стартовой площадки" (например, главного экрана) и шаг за шагом приводит его к решению поставленной задачи. При этом пользователя необходимо предупреждать о возможной опасности и каждый раз сообщать о том, что должно произойти после выполнения очередного действия (например, "откроется окно менеджера файлов").

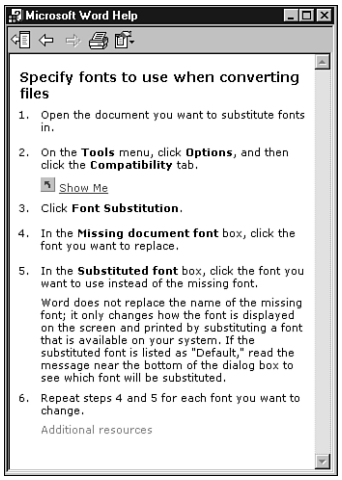


Рис. 27.1. Пример процедурной справки из приложения Microsoft Word

Однако хорошие процедурные справки составлять не так-то просто. Как узнать, какие именно шаги следует включить? Каков уровень знаний пользователя? Например, стоит ли ему напоминать, что нужно щелкнуть на кнопке Закрыть, чтобы закрыть диалоговое окно? К сожалению, однозначного ответа на этот вопрос не существует. Можно всю информацию подавать в двух вариантах: и для новичка и для опытного специалиста, но в этом случае нужно соответствующим образом ее обозначить. Можно описывать последовательность действий в расчете на профессионала, но информацию для новичка можно поместить в дополнительные окна, которые появятся, если к ним специально обратиться.

Выделяют два основных типа процедурной информации: выполнение задач и выявление неисправностей. Если инструкции первого типа помогают пользователю использовать возможности приложения для решения различных задач, то инструкции второго типа — просто бесценны в ситуациях, когда что-то не ладится.

Несколько советов для составления описания последовательности действий.

- Начните с простой формулировки назначения процедуры. Например: "Эта процедура используется для добавления в базу данных нового клиента".
- В поле зрения необходимо держать всю проблему целиком. Если пользователь, прежде чем добавлять нового клиента в базу данных, должен сначала описать его, создайте ссылку на эту тему.
- Информируйте пользователя о том, что его ожидает после выполнения действия. Если оно таит в себе определенную опасность и может вызвать потерю данных или сбой в работе системы, сообщите об этом до выполнения действия, а не после.
- Предоставьте пользователю информацию в объеме, достаточном для выполнения задачи.

Справочная информация

К этой категории относится информация, которую необязательно запоминать. К такой информации обращаются по мере необходимости (рис. 27.2). Поэтому к организации раздела справочной информации должны быть применены четкие правила, ведь главное здесь — обеспечить простоту поиска нужных сведений и переходов из одного раздела в другой. Справочная информация, как правило, не ориентирована на выполнение определенной задачи. Хороший справочник должен быть исчерпывающим; он должен содержать все темы, по которым возможны запросы. Если справочник представляет собой словарь терминов, используемых в вашем программном продукте, ни один такой термин не должен быть упущен. Например, к справочной информации относятся формулы, используемые в программе, таблицы данных и синтаксис языка программирования.

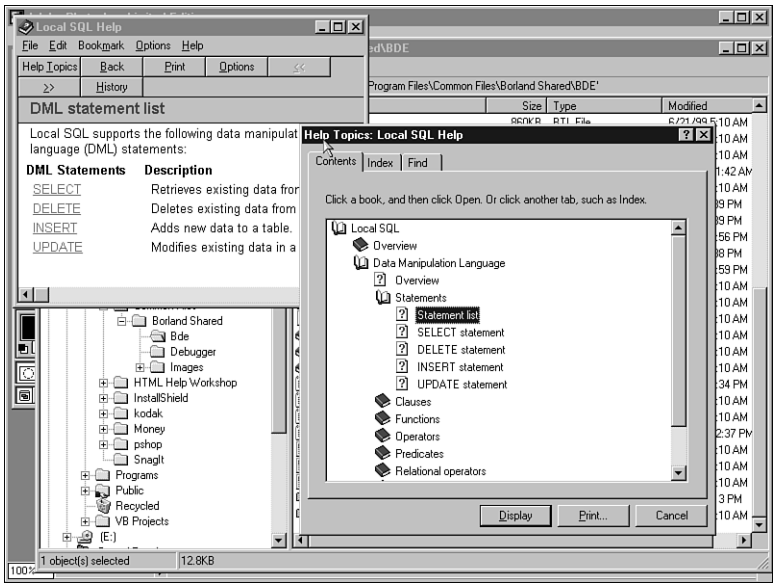


Рис. 27.2. Пример справочной информации из справочной системы Local SQL Server

Концептуальная информация

Концептуальная информация призвана ответить на все потенциальные “зачем” и “почему” относительно того, что делает ваше приложение. Эти сведения обычно не требуются при выполнении отдельных задач, но они могут быть весьма полезны при длительном использовании или поддержке программного продукта (рис. 27.3). Концептуальная информация призвана помочь пользователю в принятии решений и указать направление действий, когда такое решение уже принято. Этот тип информации не претендует на полноту — вы не обязаны включать сюда всю “подноготную” приложения, но следует предоставить пользователю достаточную информацию, чтобы у него сложилось правильное представление о том, зачем необходимо решать те или иные задачи.

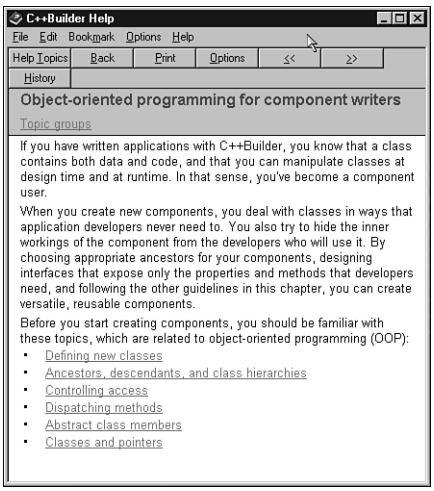


Рис. 27.3. Пример концептуальной информации, предоставляемой в C++Builder

Учебная информация

Назначение учебной информации — обучить пользователей работать с приложением. Поэтому в этой части справочной системы часто разъясняют, почему пользователь должен выполнить определенное действие и приводят толкование используемых терминов (рис. 27.4). Каждый раздел учебной информации должны включать краткое уведомление о том, что узнает пользователь, прочитав этот раздел. Не следует забывать о примерах, причем подробно описанных и прокомментированных. Чем ближе пример будет подходить к ситуации пользователя, тем лучше пойдет процесс освоения. В отличие от процедурной справки, в обучающих разделах можно не экономить на словах и примерах.

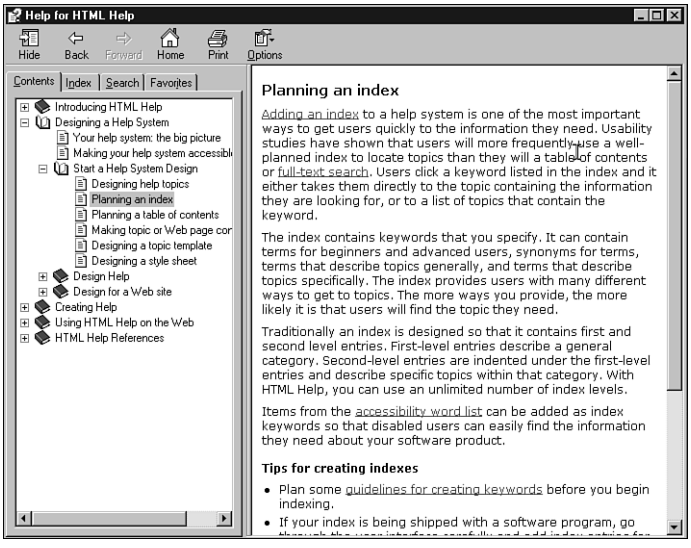


Рис. 27.4. Пример учебной информации из HTML Help Workshop

Форматы справочной информации

Справочная система, или просто справка, представляет собой документацию на программный продукт, отформатированную для прочтения на экране. В табл. 27.1 приведен список известных справочных форматов.

Таблица 27.1. Форматы справочной информации

Название	Описание	Комментарии
WinHelp, Windows Help	Компилируемый справочный формат, разработанный Microsoft (текущая версия: 4), поддерживается в Windows 95/98	Оригинальная и до сих пор самая популярная справочная система Windows, версия 4, поддерживает 256-цветную графику, содержание, алфавитный указатель, поиск текста, временные рабочие окна, работу с ключевыми словами и родственными ключевыми словами, а также гипертекстовые ссылки. Выпуск новых версий не планируется (мертвая технология), но сторонние производители поддерживают различные усовершенствования (фоновая графика, Web-связи, ускоренный поиск). Компилируется из исходного `.rtf`- в `.hlp`-формат
HTML Help	Компилируемый справочный формат, разработанный Microsoft (текущая версия: 1.3), поддерживается в Windows 98, 2000	Поддерживаемая в настоящее время справочная система пока не обладает такими функциональными возможностями, система WinHelp. Использует броузер, поставляемый вместе с Windows 98, 2000, NT5; позволяет просмотр в Windows 95 и NT4 с помощью Internet Explorer (по крайней мере с применением версии 4). Компилируется из исходного `.html`- в `.chm`-формат
NetHelp	Собственный формат HTML Help, разработанный компанией Netscape. Текущая версия: 2	Net Help никогда по-настоящему не воспринимали как формат справочной системы, тем не менее он используется в Navigator. Для NetHelp 2 необходимо использование Netscape Navigator 4 или старшей версии. Поддерживает определенную контекстную чувствительность
Независимые собственные форматы HTML Help	Собственные форматы HTML Help, разработанные сторонними разработчиками инструментов создания справочных систем	Как правило, эти форматы используют JavaScript и Java для имитации возможностей справочной системы HTML Help, но обычно они не зависят от платформы и брооузера. Примеры: WebHelp (разработан фирмой RoboHelp) и InterHelp (разработан фирмой ForeHelp)
Java Help	Справочный формат HTML/XML, разработанный компанией Sun Microsystems	Требует использования броузеров JVM и Java Help. Лучше всего подходит для Java-приложений. Работает на любой платформе

Существует множество других справочных форматов, которые непосредственно неприменимы к приложениям C++Builder (например Windows CE Help, Oracle Help и разновидности справочных форматов, пригодных для систем Linux и UNIX).

Справочные файлы в формате WinHelp: стандарт Windows

Несмотря на существование множества справочных форматов, стандарт WinHelp, благодаря разнообразию средств и простоте развертывания, — лучший вариант для большинства проектов C++Builder. Он может работать во всех системах Windows (16-разрядная версия 3 обладает обратной совместимостью с Windows 3.1), при этом существует немало бесплатных и коммерческих инструментальных средств, которые синтезируют этот формат. Следующем в списке популярности стоит формат HTML Help, который является стандартом для Windows 98, 2000 и NT5. Он также представлен в пакете MS Office 2000. Самая последняя версия формата HTML Help (1.3, выпущенная с Windows 2000) обладает массой возможностей, но более сложна для инсталляции в том случае, если вы не намерены ограничиться одной операционной системой (Windows 2000). Например, HTML Help требует использования по крайней мере Internet Explorer (IE) 4.

Основное отличие формата WinHelp от HTML Help заключается в степени поддержки в среде C++Builder. Все свойства и методы, связанные со справочными материалами в C++Builder, разработаны в расчете на формат WinHelp и практически не требуют дополнительного программирования для обеспечения контекстной чувствительности. С другой стороны, формат HTML Help слабо поддерживается в C++Builder, но может быть реализован при затрате дополнительных усилий.

Рассмотрим возможности стандарта WinHelp (рис. 27.5) и некоторые инструменты, предназначенные для его создания.

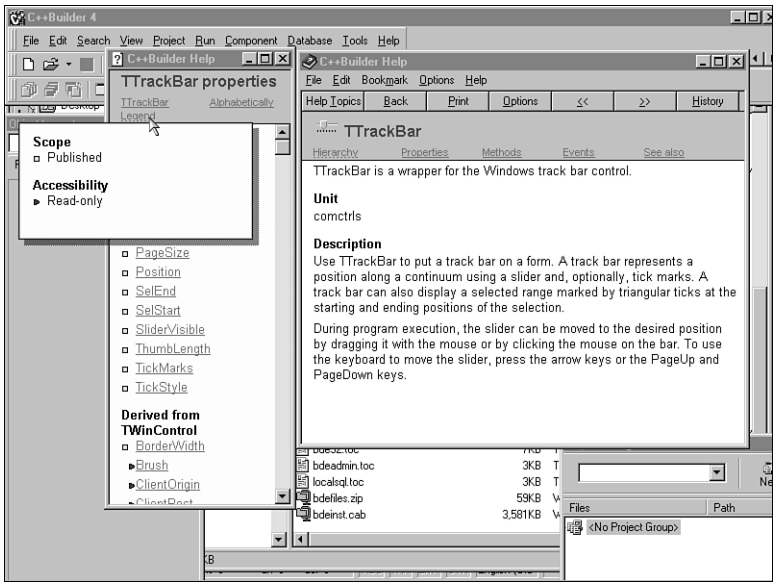


Рис. 27.5. Пример стандартной WinHelp-системы для C++Builder. Вы видите несколько окон: главное, вторичное и всплывающее окно

Ниже описаны строительные блоки WinHelp.

- *Раздел*. Это раздел текста, посвященный одной теме. Разделы имеют заголовки и отображаются в окнах (окна могут иметь разные области: с полосами прокрутки и без

них, но заголовок чаще всего отображается в верхней части стационарной, т.е. непрокручиваемой, области).

- *Ссылки*. Они также именуются переходами и представляют собой гипертекстовые ссылки, которые позволяют перейти к другому разделу. В WinHelp гиперссылки всегда подчеркиваются одной чертой и окрашены в зеленый цвет. Всплывающие ссылки предназначены для отображения тематического раздела внутри другого всплывающего окна, которое исчезает после щелчка мышью. Всплывающие ссылки всегда подчеркиваются пунктирной линией и окрашены в зеленый цвет.
- *Содержание*. Это список основных тем, включенных в состав справочного файла. Содержание оформляется в виде дерева просмотра "книг" (с одним или несколькими разделами) и "подкниг". Содержание может включать элементы справочных файлов и ссылаться на связанные справочные разделы.
- *Алфавитный указатель*. Список ключевых слов, которые ссылаются на связанные с ними разделы.
- *Средства поиска*. WinHelp поддерживает поиск по полному тексту. Индекс поиска динамически генерируется при первом использовании справочного файла (в `.fts`-файле).

Инструментальные средства создания справочных систем

На профессиональном поле инструментальных средств создания справочных систем не так уж много игроков. В табл. 27.2 перечислены самые популярные коммерческие продукты.

Таблица 27.2. Инструментальные средства создания справочных систем

Продукт	Изготовитель	Комментарии
RoboHelp	Bluesky	Полный набор инструментальных средств, выпускаемый в различных вариантах. Опирается на MS Word. Создает форматы HTML Help, WinHelp и ряд других
ForeHelp	ForeHelp	Полный набор инструментальных средств. Распространяется в версиях HTML и WinHelp. Самодостаточный редактор
Doc-To-Help	WexTech	Полный набор инструментальных средств. Опирается на MS Word
WebWorks Publisher	Quadralay	Полный набор инструментальных средств. Дорогостоящий. Опирается на Adobe FrameMaker. Создает форматы WinHelp, HTML и ряд других

Контекстно-зависимая справка

Контекстно-зависимой, или контекстно-чувствительной, называется справка, которая "держит нос по ветру", т.е. находится в курсе того, какой элемент управления или форма пребывает в активном состоянии. Это значит, что, когда пользователь работает с диалоговым окном, курсор стоит в текстовом поле или указатель мыши "завис" над какой-нибудь кнопкой, появляется справка, связанная с данным элементом управления. Обычно различают четыре типа контекстно-зависимой справки:

- всплывающие подсказки;
- "Что это такое?", или всплывающие справочные окна;
- стандартный файл WinHelp;
- встроенная справка.

Всплывающие подсказки

Всплывающие подсказки появляются, когда указатель мыши замирает над каким-нибудь элементом управления на несколько секунд. Текст этих подсказок обычно незатейлив и описывает назначение элемента или даже просто называет его.

„Что это такое?“, или всплывающие справочные окна

Этот тип справки появляется, когда пользователь щелкает на элементе правой кнопкой мыши или сначала щелкает на кнопке What's This? (Что это такое?), а затем — на интересующем его элементе управления (рис. 27.6). В результате появляется всплывающее окно, в котором обычно содержится описание элемента, которое порой дополняется кратким перечнем действий, направленных на выполнение конкретной задачи.

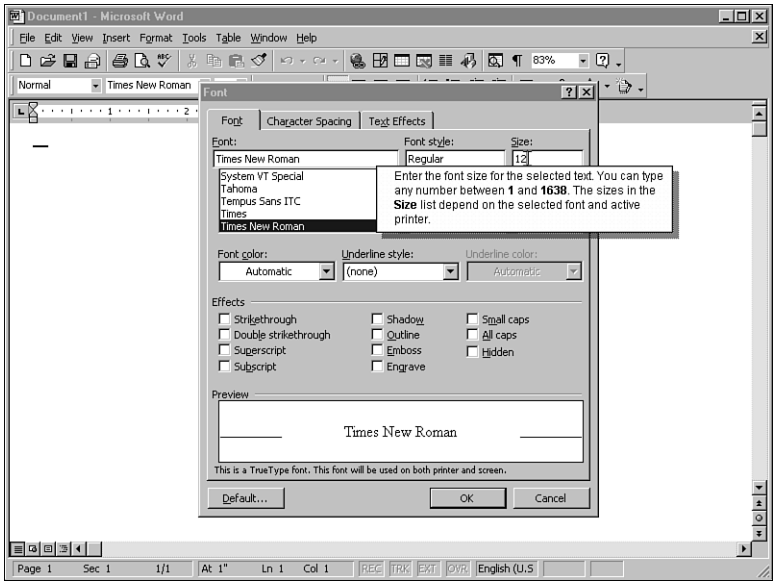


Рис. 27.6. Пример справки “Что это такое?”

Стандартный файл WinHelp

Когда пользователь нажимает клавишу <F1>, C++Builder открывает справочный файл приложения. Если идентификатор (ID) контекста связан с формой или активным элементом управления, то после нажатия клавиши <F1> отображается тематический раздел справки, соответствующий данному ID контекста. Это прекрасный способ представить обзор задач, которые можно выполнить, используя данную форму или элемент управления.

Встроенная справка

К этой категории справочной информации относится текст, который чаще всего “попадается на глаза” пользователям (в диалоговых окнах, в окнах сообщений об ошибках, на кнопках и других элементах управления), т.е. сюда включается вся информация справочного характера, которая жестко встраивается в код приложения.

Компилятор MS Help Workshop

C++Builder поставляется вместе с WinHelp-компилятором Help Workshop компании Microsoft. Его файл (`hcw.exe`) находится в C++Builder-папке `/help/tools/`. Вы можете писать простые текстовые файлы в формате WinHelp, не используя ничего, кроме текстового редактора, который способен сохранить файл в формате так называемого обогащенного текста (`.rtf`), и компилятора `hcw.exe`. Однако этот метод не рекомендуется применять для создания больших справочных систем, в которых предполагается реализовать много возможностей версии WinHelp 4. Такой подход можно сравнить с программированием вручную большого Web-узла на языке HTML; это, безусловно, возможно, но очень утомительно и чревато предрасположенностью к ошибкам. Лучше всего воспользоваться одним из бесплатных, условно-бесплатных или коммерческих продуктов (см. раздел "Инструментальные средства создания справочных систем, доступные в Internet" в конце этой главы). Прочитав упомянутый раздел, вы узнаете, насколько просто автоматизировать этот процесс с помощью нескольких макросов в редакторе текстовых сценариев (таком как Word), который использован во многих коммерческих и некоммерческих пакетах. И в самом деле, в большинстве систем создания справочных материалов в качестве компилятора используется `hcw.exe`. После обработки кодов тематических разделов, элементов указателя и пр. предварительно скомпилированная версия справочной системы сохраняется в `.rtf`-формате, который затем компилируется программой `hcw.exe`.

Данный раздел можно считать небольшим самоучителем, благодаря которому вы сможете приступить к работе с компилятором MS Help Workshop. Для получения более подробной информации о функциях интерфейса WinHelp API и возможностях самого компилятора можно также заглянуть в справочную систему HCW. Файлы примеров, используемые в самоучителе, находятся на прилагаемом к книге компакт-диске (в папке `HelpExample`). Файл в формате обогащенного текста называется `help_ex.rtf`, файл карты распределения — `help_ex.map`, файл проекта — `help_ex.hpj`, а скомпилированный файл носит имя `help1.hlp`. Сюда также включен проект C++Builder, который вызывает этот справочный файл.

Справочный файл WinHelp компилируется из исходного файла, сохраненного в формате обогащенного текста, но он опирается на теги, используемые для создания отдельных справочных разделов, ID разделов, связей между разделами, элементов указателя и последовательности просмотра. Теги вставляются в виде сносок на первой строке раздела. Формально формат WinHelp для компиляции требует только наличия ID разделов. В действительности для справочной системы требуется по меньшей мере задание последовательности просмотра, заголовков разделов и ссылок.

Ниже приводится описание действий, необходимых для создания простой справочной системы к проекту C++Builder.

1. Создайте файл тематических разделов справочной системы. (В данном примере я использую редактор MS Word, но можно использовать любой текстовый редактор, который способен сохранять файл в формате обогащенного текста.)

   Откройте новый файл. Напишите список разделов, причем название каждого раздела должно быть написано на отдельной строке. Под названием раздела напишите несколько предложений, образующих справочную информацию. Первым должен быть раздел *Обзор* или *Содержание*, поскольку этот простой справочный файл не имеет стандартной вкладки Topics (Разделы).

   Overview (Обзор)

   Edit Box (Поле редактирования)

   Check Box (Флажок)

   Glossary (Словарь терминов)

На рис. 27.7 этот файл показан в окне Microsoft Word после добавления сносок.

2. Каждый раздел должен включать теги для индикации ID, заголовка и пр. Введите тег, разместив курсор перед текстом заголовка раздела, и вставьте сноску. Для символа сноски используйте соответствующий символ тега. В саму сноску введите ID, заголовок, ключевые слова или текст последовательности просмотра. Теги для вставки в каждый раздел приведены в табл. 27.3.

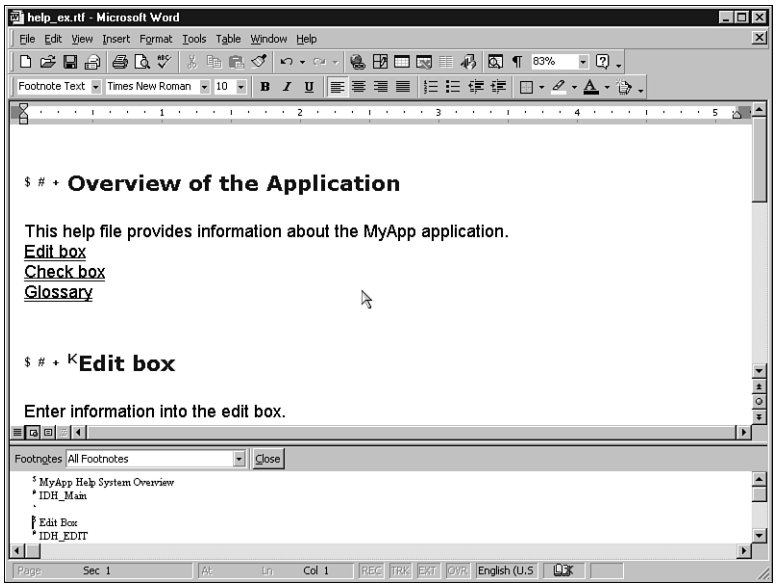


Рис. 27.7. Пример редактирования файла разделов в Microsoft Word

Таблица 27.3. Теги разделов

Тип тега	Символ	Назначение	Примечание
ID	#	Каждому разделу нужен уникальный идентификатор. Используется в качестве имени приемника для ссылок	Не используйте следующие символы: `# = + @ * % !`. Используйте не более 255 символов. Если раздел будет вызываться из C++-приложения, идентификатор должен начинаться с префикса `IDH_` (контекстно-чувствительный)
Заголовок	$	Заголовок раздела. В таком же виде он будет отображен во вкладке Contents (Содержание) и в окнах Find (Найти)	До 255 символов (с учетом пробелов)
Ключевое слово	K	Список ключевых слов, используемых для создания указателя справочного файла	Ключевые слова должны быть разделены точкой с запятой. До или после ключевых слов не используйте пробелы. Не используйте символы возврата каретки. Для одного ключевого слова разрешено не более 255 символов

Окончание табл. 27.3

Тип тега	Символ	Назначение	Примечание
Ключевое слово A	A	Список ключевых слов, используемых для связи данного раздела с другими. Используется в основном для кнопок See Also (См. также)	Ключевые слова должны быть разделены точкой с запятой. До или после ключевых слов не используйте пробелы. Не используйте символ возврата каретки. Для одного ключевого слова разрешено не более 255 символов
Последовательность просмотра	+	Используется для определения порядка разделов, когда пользователь щелкает на кнопках просмотра (<< и >>). В большинстве систем используется единый порядок просмотра	Не используйте символы # = +@ * % !, а также пробелы. Используйте не более 50 символов. Для применения последовательностей просмотра необходимо также активизировать кнопки просмотра (Browse) в файле проекта. Самый простой способ активизировать автоматический порядок просмотра — добавить тег + в каждый раздел, но оставить его пустым. HCW поместит разделы в порядке их следования в исходном файле; для изменения порядка достаточно изменить порядок разделов в .rtf-файле
Окно	>	Используется для определения типа окна, в котором будет отображаться текст раздела. Стандартные системы используют три типа окон: для основных разделов, для описания последовательности действий и всплывающие окна (или окна для справки типа “Что это такое?”)	Используйте то же имя, которое определено для окна в файле проекта справки (см. п. 7). По умолчанию используется главное окно. Этот тег не нужно вводить для разделов, отображаемых в главном окне

3. Определите параметры форматирования текста для проекта справочной системы: шрифт, цвет и размер шрифта. При форматировании текста не используйте шрифты, которые могут быть не установлены в системе пользователя (другими словами, используйте только стандартные шрифты Windows). Что касается таблиц, то их перевод в WinHelp нельзя назвать удачным, поскольку WinHelp-таблицы не повторяют заливку колонки текстом, а такие параметры, как тип границы и затенение вообще не переносятся в формат WinHelp. Если все-таки необходимо использовать таблицу, сделайте ее как можно *у*же и используйте простой тип границы. Для переноса затенения и специальных атрибутов форматирования сохраните таблицу в виде графического .bmp-файла и отобразите ее соответствующим образом.

4. Создайте переходы к тематическим разделам.

   Выполните следующие действия для создания внутрисправочной системы гипертекстовых переходов из одного раздела в другой.

   a. Найдите текст (или графическое изображение), который вы собираетесь отметить как переход.

      Пример. За дополнительной информацией обращайтесь к статье “Создание электронной справочной системы”.

b. Введите ID для раздела, на который выполняется переход, после текста ссылки (без дополнительного пробела).

   Пример (если ID раздела равен значению `IDH_WRITING_HELP`). За дополнительной информацией обращайтесь к статье “Создание электронной справочной системы” `IDH_WRITING_HELP`.

c. Выделите текст перехода и отформатируйте его с помощью подчеркивания двойной линией (рис. 27.8). (Форматирование подчеркиванием одинарной линией используется для создания разделов, отображаемых во всплывающих окнах.)

d. Выделите ID раздела и отформатируйте его в виде скрытого текста.

   Пример. За дополнительной информацией обращайтесь к статье “Создание электронной справочной системы”.

5. Сохраните Word-файл в формате обогащенного текста (`.rtf`).

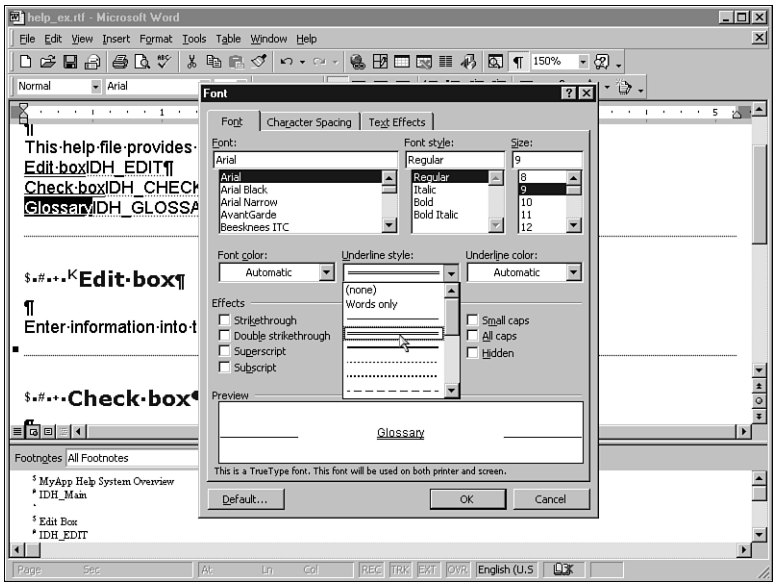


Рис. 27.8. Установка формата двойного подчеркивания в редакторе Word

6. Создайте MAP-файл, который связывает справочные разделы с Help-идентификаторами C++Builder (C++Builder Help ID). Этот файл можно создать в редакторе Word или в любом другом текстовом редакторе, поскольку это обычный текстовый файл.

Справочные разделы можно связать с какой-либо формой или свойством `HelpContext` VCL-компонента. Самый простой способ назначения справочных разделов различным компонентам — установить свойство `HelpContext` каждого компонента во время разработки, а затем установить соответствие свойства `HelpContext` и ID разделов в MAP-файле.

Существуют и другие способы вызова конкретных справочных разделов из среды C++Builder — см. раздел “Добавление в C++Builder справки “Что это такое?” ” ниже в этой главе.

Если вы хотите вручную добавлять в справочный проект пары Topic ID/Context ID (ID раздела/ID контекста), можно опустить создание отдельного файла карты распределения (MAP-файла). Такой вариант подойдет для небольших проектов, но обширными проектами проще управлять, используя один или даже несколько MAP-файлов. Например, разработчики могли бы создать по отдельному MAP-файлу для каждой формы. В обновлениях пришлось бы редактировать только файлы, связанные с измененными формами.

MAP-файлы представляют собой текстовые файлы ASCII, каждая строка в которых имеет следующий формат.

```
#define TOPIC_ID HelpContextNumber
```

Здесь `TOPIC_ID` — ID раздела, а `HelpContextNumber` — свойство `HelpContext` элемента управления VCL. В MAP-файлах применим C/C++-стиль комментариев. В следующем фрагменте программы показан пример содержимого MAP-файла.

```
/* MyAppHelp.MAP
   В главной форме используются следующие тематические разделы.
*/

#define IDH_OVERVIEW 10 // обзорный раздел
#define IDH_FORM1 20    // раздел по умолчанию формы 1
#define IDH_EDIT1 30    // раздел поля редактирования 1
#define IDH_CHECK1 40   // раздел флажка 1
#define IDH_FORM2 50    // раздел по умолчанию формы 2
#define IDH_EDIT2 60    // раздел поля редактирования 2
```

Сохраните MAP-файл в формате текстового файла. Придайте ему расширение `.map`.

7. Создайте файл проекта справочной системы и скомпилируйте справочный файл.

Запустите компилятор Help Workshop (рис. 27.9).

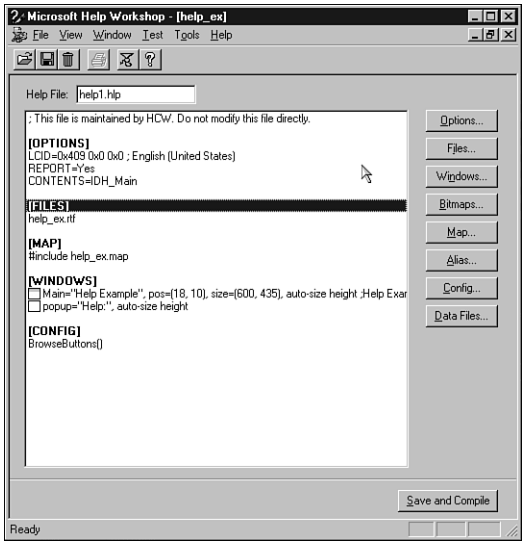


Рис. 27.9. Главный экран компилятора Microsoft Help Workshop

Добавьте в проект исходный `.rtf`-файл, щелкнув на кнопке Files (Файлы). Откроется диалоговое окно Topic Files (Файлы тематических разделов). Щелкните на кнопке Add (Добавить) и выберите вариант `.rtf`.

Чтобы добавить кнопки Browse (Просмотр), добавьте окно справки, а затем и сами кнопки. Для этого щелкните на кнопке Windows (Окна) и введите имя для главного окна (обычно используется имя `MAIN`). Для второго окна добавьте тип всплывающих окон (для справки "Что это такое?") — отнесите это окно к разряду "error message" (т.е. сообщение об ошибке). Установите такие свойства, как текст строки заголовка, позиция (для простоты используйте автоматическую регулировку размеров), цвет и кнопки. Во вкладке Buttons (Кнопки) установите опцию Browse.

Добавьте в проект MAP-файл. Для этого щелкните на кнопке Map (Соответствие), затем — на кнопке Include (Включить) и добавьте `.map`-файл. Обратите внимание, что ID разделов и идентификационные номера контекстов можно добавлять вручную (после щелчка на кнопке Add). Этот способ можно расценивать как альтернативный созданию MAP-файла (в п. 5).

Щелкните на кнопке Save and Compile (Сохранить и скомпилировать), чтобы сохранить проект и скомпилировать справочный файл.

8. Свяжите справочный файл со своим C++Builder-проектом.

В своем проекте установите этот скомпилированный справочный файл как справочный файл приложения, выбрав для этого команду Project⇨Options⇨Application⇨Help file (Проект⇨Параметры⇨Приложение⇨Справочный файл), или установите справочный файл во время выполнения приложения с помощью следующего кода.

```
Application->HelpFile = "MyApp.hlp";
```

Затем установите свойство `HelpContext` каждого компонента равным числу, используемому в справочном файле. Например, установите свойство `HelpContext` формы 1 (`Form1`) равным числу 10. Компоненты с нулевым значением свойства `HelpContext` наследуют значение того же свойства родительского компонента.

Теперь можно скомпилировать и протестировать справочный файл приложения. Нажмите клавишу <F1>, когда любой из элементов управления получит фокус, чтобы просмотреть соответствующий ему справочный раздел.

Добавление справки „Что это такое?“

Справка "Что это такое?" особенно подходит к диалоговым окнам, которые позволяют пользователю выполнить определенную задачу и в которых пользователь может потребовать справку по каждому элементу управления. Обычно доступ к такой справке реализуется с помощью кнопки с изображением вопросительного знака (?), расположенной в строке заголовка окна. Когда пользователь щелкает на такой кнопке, курсор приобретает форму вопросительного знака за счет того, что свойство `TCursorType` устанавливается равным значению `crHelp` (`-20`); затем, когда пользователь щелкает на каком-нибудь элементе управления, появляется всплывающее окно, содержащее текст справочного раздела, связанного с идентификационным номером свойства `HelpContext` этого элемента управления. Создание справки "Что это такое?" аналогично созданию стандартной справки; при этом стоит обратить внимание на следующие моменты.

- В разделы справки "Что это такое?" не следует включать область без полосы прокрутки. Если вы все-таки решили ее включить, в ней будет отображаться только текст. Об-

ласти без полосы прокрутки любого справочного окна определяются с помощью тега абзаца “keep with next” (т.е. *придерживаться следующего*). Весь текст (в начале раздела), к которому применен этот тег, будет выведен в области без полосы прокрутки.

- В C++Builder установите форму, для которой вы бы хотели предоставить справку “Что это такое?” со следующими свойствами.
  - Для свойства `BorderStyle` следует установить значение `bsDialog`. Это делает форму стандартным диалоговым окном с рамкой, размеры которой нельзя изменять.
  - Для свойства `BorderIcons` следует так установить следующие его подсвойства: `biSystemMenu = true`, `biMinimize = false`, `biMaximize = false`, `biHelp = true`.

Формы с такими параметрами будут иметь кнопку Что это такое?, а текст справки будет выводиться во всплывающем окне в позиции курсора. Использование этих параметров иллюстрируется на примере диалогового окна, показанного на рис. 27.10. Это диалоговое окно относится к справочному проекту, приведенному на прилагаемом к книге компакт-диске.

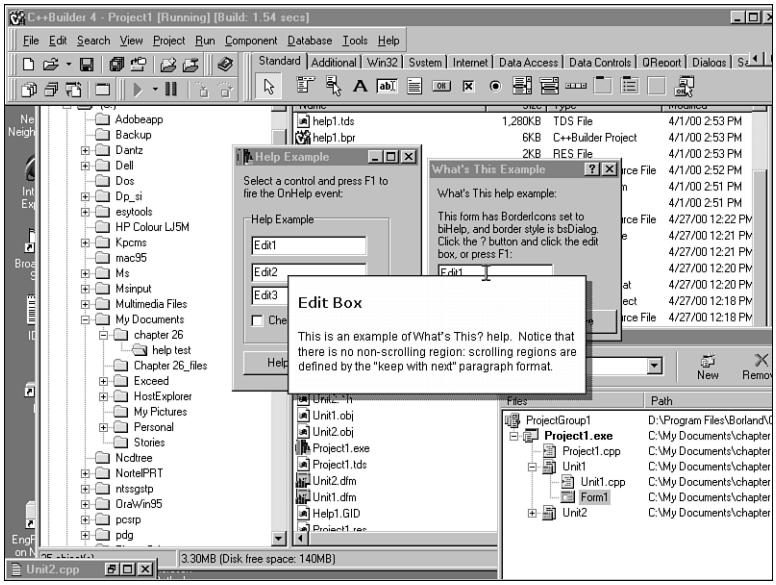


Рис. 27.10. Справка “Что это такое?” в C++Builder-форме

Одно предостережение. Если не предпринять никаких мер, нажатие клавиши <F1> приведет к отображению справки во всплывающем окне. Поэтому, чтобы справка отображалась в стандартном WinHelp-окне, используйте в обработчике события `OnHelp` методы `Application->HelpJump()` или `Application->HelpContext()`.

Расширение справочной системы с помощью более сложных средств разработки

WinHelp поддерживает множество других средств, которые можно использовать для усовершенствования вашей справочной системы.

- *Макросы*. WinHelp поддерживает большое количество макросов, расширяющих функции справочной системы. Например, чтобы добавить ссылку на разделы, связанные с

печатью (*print*), введите текст самой гиперссылки, а затем — восклицательный знак и макрос `KLink`: `related topics!KLink(printing, print, printer)`. Примените к тексту гиперссылки подчеркивание двойной линией, а к тексту макроса форматирование, скрывающее символы (включая восклицательный знак). Microsoft Help Workshop содержит полную информацию о макросах WinHelp.

- *Тег построения*. Для тематических разделов можно задать тег построения, а затем в файле проекта справочной системы указать, какие теги построения включить, а какие — исключить из справочного файла. Это позволит создать единственный исходный файл справки для различных версий одного приложения. Например, вы могли бы присвоить тег построения `pro` определенным разделам, которые применяются только в профессиональной версии приложения. Достаточно исключить тег построения `pro` при компиляции справочного файла для базовой версии, и эти разделы не будут включены в конечный файл.
- *Тип окна*. WinHelp позволяет определить несколько различных типов окон. Обычно стандартное окно раздела содержит область заголовка без полосы прокрутки и прокручиваемую область, содержащую текст раздела. Возможно, вы решите определить второе окно для описания последовательности действий и еще одно окно для справочной информации во всплывающем окне. Например, справка "Что это такое?" не может содержать непрокручиваемую область.

Справочные файлы Microsoft HTML Help

В отличие от компилятора Help Workshop (который поставляется в составе C++Builder), для работы с компилятором Microsoft HTML Help Workshop самую последнюю его версию необходимо загрузить с Web-узла по адресу: `http://msdn.microsoft.com/workshop/author/htmlhelp/`. Его окно показано на рис. 27.11.

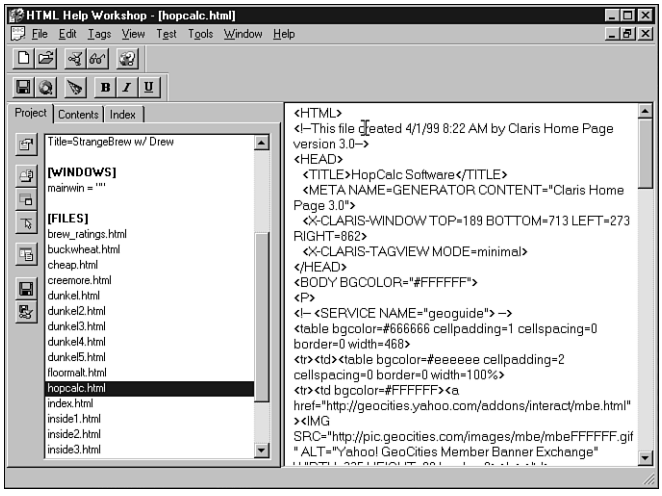


Рис. 27.11. Программа просмотра Microsoft HTML Help Workshop

Для работы компилятора HTML Help требуется броузер Internet Explorer (хотя бы версии 4) в среде Windows 95 или NT4. Броузер Internet Explorer, который используется в качестве программы просмотра файлов HTML Help, поставляется с Windows 98, 2000 и NT5.

HTML Help — это компилируемый формат, который по параметрам сжатия зарекомендовал себя так же хорошо, как формат zip-файлов. И в самом деле с его помощью можно упаковывать большие Web-узлы и распространять их в единой компилируемой форме. HTML Help Workshop также предлагает возможность перекомпиляции исходного WinHelp-формата (разделы — в `.rtf`-файле, содержание — в `.cnt`-файле, а проект — в `.hpj`-файле) в формат HTML Help (`.chm`-файл с содержанием). Формат HTML Help предлагает некоторые интересные визуальные усовершенствования: панель автоматического поиска/перехода по разделам, которая появляется, когда указатель мыши располагается поверх нее, гиперссылки, которые реагируют на события, связанные с мышью, и способность бесшовно переходить из справки HTML Help в пространство Web (рис. 27.12). Поскольку этот формат справочных файлов использует Internet Explorer в качестве программы просмотра, в справочную систему можно включить JavaScript и VBScript.

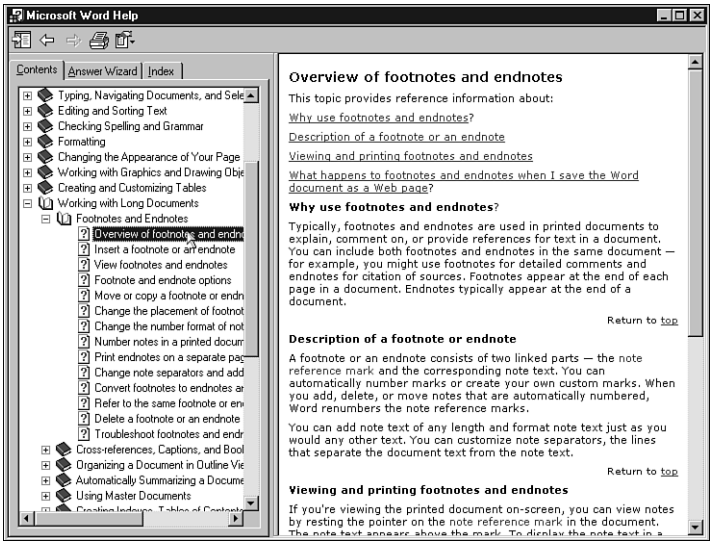


Рис. 27.12. Пример справочной системы в формате HTML Help для Microsoft Word

Учитывая все вышесказанное, следует рассмотреть создание справочной системы в формате HTML Help в следующих случаях.

- У вас имеется большой объем документации, которая уже создана в формате HTML (например, документация на программное обеспечение в Web).
- Вы собираетесь установить Windows 98 или 2000 и ничего не имеете против Internet Explorer.
- Вы не прочь поработать над созданием дополнительного программного кода, необходимого для ввода функций интерфейса HTML Help API в проект C++Builder.

Свойства и методы VCL-компонентов справочной системы

C++Builder позволяет легко интегрировать справочные системы в формате WinHelp и приложения. Ниже описаны свойства и методы компонентов VCL, которые используются при развертывании и настройке справочной системы.

Свойства компонентов справочной системы

TWinControl::HelpContext

Свойство `HelpContext` — это строительный кирпичик контекстно-зависимой справки. Когда элемент управления получает фокус и пользователь нажимает клавишу <F1>, отображается раздел справки, связанный со значением свойства `HelpContext` этого элемента управления. Если значение свойства `HelpContext` равно `0`, значит, этот элемент наследует значение свойства `HelpContext` родительского элемента управления. Таким образом, одно и то же значение свойства `HelpContext` могут иметь все элементы в форме или все элементы в группе (GroupBox).

TApplication::HelpFile

Это свойство устанавливает по умолчанию справочный файл для приложения. Установите это свойство во время разработки в поле Help File (Справочный файл), расположенном во вкладке Application (Приложение) диалогового окна Project Options (Параметры проекта). Это свойство можно установить во время работы приложения, чтобы динамически изменить справочный файл, связанный с приложением (например, для изменения языка).

TCustomForm::HelpFile

Используйте это свойство, чтобы назначить форме другой справочный файл. Это очень удобно, если вы разрабатываете несколько приложений, которые используют одни и те же формы. Разработайте справочные разделы для такой "общей" формы и поместите их в отдельный справочный файл, а затем назначьте его этому свойству. Когда пользователь нажмет клавишу <F1>, работая с этой формой, будет отображен ее собственный справочный файл.

Методы компонентов справочной системы

TApplication::HelpCommand

Этот метод позволяет получить доступ к интерфейсу WinHelp API непосредственно из C++Builder. Список команд интерфейса WinHelp API приведен в табл. 27.4. В этом примере вызывается команда `HELP_CONTENTS`, которая отображает содержание справочного файла.

```
bool __fastcall HelpCommand(int Command, int Data);
```

Пример.

```
Application->HelpCommand(HELP_CONTENTS, 0);
```

Таблица 27.4. Методы интерфейса WinHelp API

Команда	Действие	Данные
HELP_COMMAND	Выполняет макрос (макросы) справочной системы	Адрес строки, которая задает имя Help-макроса (макросов), подлежащих выполнению. Если в строке задано несколько имен макросов, они должны быть разделены двоеточием или точкой с запятой. Для имени макроса нужно использовать короткую форму, поскольку формат WinHelp не поддерживает длинные имена
HELP_CONTENTS	Отображает тематический раздел, заданный опцией Contents в секции *OPTIONS* .HPJ-файла. Эта команда используется с целью обратной совместимости. В новых программах следует предоставить .CNT-файл и использовать команду HELP_FINDER	Игнорируются; установлены равными 0
HELP_CONTEXT	Отображает раздел, идентифицированный заданным идентификатором контекста в секции *MAP* .HPJ-файла	Длинное целое значение без знака, содержащее идентификатор контекста для раздела
HELP_CONTEXTMENU	Отображает меню Help для выбранного окна, а затем отображает (во всплывающем окне) раздел для выбранного элемента управления	Адрес массива пар слов двойной длины. Первое слово в каждой паре — идентификатор, а второе — числовое значение контекста для раздела
HELP_CONTEXTPOPUP	Отображает во всплывающем окне раздел, идентифицируемый заданным идентификатором контекста, определенным в разделе *MAP* .HPJ-файла	Длинное целое значение без знака, содержащее идентификатор контекста для раздела
HELP_FINDER	Отображает диалоговое окно Help Topics (Разделы справки)	Игнорируются; установлены равными 0
HELP_FORCEFILE	Гарантирует, что WinHelp отображает корректный справочный файл. Если будет отображаться некорректный файл, WinHelp откроет корректный; в противном случае никаких действий выполнено не будет	Игнорируются; установлены равными 0
HELP_HELPONHELP	Отображает справочную информацию о том, как использовать формат WinHelp, если доступен файл WINHLP32.HLP	Игнорируются; установлены равными 0
HELP_INDEX	Отображает раздел, заданный опцией CONTENTS в разделе *OPTIONS* .HPJ-файла. Эта команда используется с целью обратной совместимости. В новых программах следует предоставить .CNT-файл и использовать команду HELP_FINDER	Игнорируются; установлены равными 0

Продолжение табл. 27.4

Команда	Действие	Данные
HELP_KEY	Отображает раздел в таблице ключевых слов, который соответствует заданному ключевому слову, если произошло только одно точное совпадение. Если таких совпадений несколько, эта команда отображает список Topics Found (Найдены разделы)	Адрес строки, содержащей ключевые слова. Несколько ключевых слов необходимо разделить точкой с запятой
HELP_MULTIKEY	Отображает раздел, заданный ключевым словом в альтернативной таблице ключевых слов	Адрес структуры MULTIKEYHELP, которая задает символ сноски таблицы и ключевое слово
HELP_PARTIALKEY	Отображает раздел в таблице ключевых слов, который соответствует заданному ключевому слову, если произошло только одно точное совпадение. Если таких совпадений несколько, эта команда отображает диалоговое окно Topics Found (Найдены разделы). Чтобы отобразить предметный указатель без передачи ключевого слова, следует использовать указатель на пустую строку	Адрес строки, содержащей ключевые слова. Несколько ключевых слов необходимо разделить точкой с запятой
HELP_QUIT	Информирует программу WinHelp, что она больше не нужна. Если ни одна из программ не запрашивала справочную систему, Windows закрывает программу WinHelp	Игнорируются; установлены равными 0
HELP_SETCONTENTS	Задает раздел "Содержание". Программа WinHelp отображает этот раздел, когда пользователь щелкает на кнопке Contents (Содержание), если Help-файл не имеет связанного .CNT-файла	Длинное целое число без знака, содержащее идентификатор контекста для раздела оглавления
HELP_SETPOPUP_POS	Устанавливает позицию последующего всплывающего окна	Адрес структуры POINTS. Всплывающее окно располагается на экране так, как если бы курсор мыши указывал позицию его размещения в момент вызова всплывающего окна
HELP_SETWINPOS	Отображает окно Help, если оно свернуто или находится в памяти, и устанавливает его размеры и местоположение в соответствии с заданными значениями	Адрес структуры HELPWININFO, которая задает размеры и местоположение первичного либо вторичного окна Help
HELP_TCARD	Означает, что команда предназначена для экземпляра учебной карточки программы WinHelp. Программа комбинирует флаг HELP_TCARD с другими командами, используя поразрядный оператор OR (ИЛИ)	Зависит от команды, с которой сочетается флаг HELP_TCARD

Окончание табл. 27.4

Команда	Действие	Данные
HELP_WM_HELP	Отображает во всплывающем окне раздел для элемента управления, идентифицируемого значением hwnd	Адрес массива пар слов двойной длины. Первое двойное слово в каждой паре — идентификатор элемента управления, а второе — идентификатор контекста для справочного раздела

TApplication::HelpContext и TApplication::HelpJump

Эти два метода позволяют C++Builder напрямую вызывать справочные разделы WinHelp. Метод `HelpContext` использует числовое значение контекста для справочного раздела, а метод `HelpJump` — ID справочного раздела. Используемый справочный файл определяется свойством `TApplication::HelpFile`.

Примеры.

```
Application->HelpFile = "MyHelp.hlp";
Application->HelpContext(10);
Application->HelpJump ("IDH_OVERVIEW");
```

События

TApplication::OnHelp

Это событие происходит при вызове функций интерфейса WinHelp API из среды C++Builder (например, при вызове методов `HelpContext` или `HelpJump`). Напишите обработчик события `OnHelp` для специальной обработки методов справочной системы.

```
typedef bool __fastcall (__closure *THelpEvent)(Word Command,
                                                int Data, bool &CallHelp);
```

Установите для параметра `CallHelp` значение `true`, чтобы вызвать WinHelp-метод после совершения события, или значение `false`, чтобы прекратить работу с интерфейсом WinHelp.

Источники информации по созданию справочных систем

Существует множество источников информации (как в печатном, так и электронном виде), способных помочь в трудном деле написания технической документации. Лучшие из них перечислены в следующих разделах.

Книги

По написанию технической документации можно найти ряд хороших книг. Некоторые из них названы ниже. В этом списке вы найдете два наиболее популярных руководства по стилю (т.е. оформлению системной и программной документации), поскольку без такой литературы невозможно создать приличную справочную систему.

- JoAnn T. Hackos, Dawn M. Stevens, *Standards for Online Communication*, Wiley, 1997.

- William Horton, *Designing and Writing Online Documentation*, Wiley, 1994.
- Robert W. Bly, Gary Blake, *How to Write Usable User Documentation*, Macmillan, 1995.
- *Microsoft Manual of Style for Technical Publications*, Microsoft Press, 1998.
- *Read Me First! A Style Guide for the Computer Industry*. Sun Technical Publications/Prentice Hall, 1996.

Инструментальные средства создания справочных систем, доступные в Internet

Существует множество инструментальных средств создания справочных систем, и их число растет день ото дня. Даже такие промышленные лидеры, как RoboHelp и ForeHelp, предлагают пробные или упрощенные (с усеченным набором функциональных возможностей) версии в Internet. Ниже приводится список инструментальных средств создания справочных систем, которые можно найти в Internet.

- Имя: AnetHelp

  Стоимость: условно-бесплатный продукт ($149,95–$199,95)

  Системные требования: Win 95

  Web-узел: `http://www.anetsoft.com/engl/helptool/prhlptl.htm`

  Результат: Windows 3.x, Windows 95, HTML Help, Java Help
- Имя: Astrohelp

  Стоимость: бесплатный продукт

  Системные требования: Win 3.x/95, Word 6/95/97

  Web-узел: `http://www10.pair.com/vsap/AstroHelp.html`

  Результат: Windows 3.x, Windows 95
- Имя: AuthorIT

  Стоимость: коммерческая демонстрационная версия. Ограничена малым количеством тематических разделов

  Системные требования: Win 95/NT4

  Web-узел: `http://www.oscl.com/`

  Результат: Windows 95, HTML Help
- Имя: Doc-To-Help

  Стоимость: 30-дневная коммерческая демонстрационная версия

  Системные требования: Win 95/NT, Word 95/97

  Web-узел: `http://www.wextech.com/doc2help.htm`

  Результат: Windows 3.x, Windows 95, HTML Help, JavaHelp
- Имя: EasyHelp

  Стоимость: 30-дневная коммерческая демонстрационная версия

  Системные требования: Win 95/NT, Word 6/95/97

  Web-узел: `http://www.eon-solutions.com/easyhelp/easyhelp.htm`

  Результат: Windows 95, HTML Help

- Имя: eAuthor Help

  Стоимость: 30-дневная коммерческая демонстрационная версия

  Системные требования: Win 95/NT 4

  Web-узел: `http://www.hyperact.com/eAuthorHelp.html`

  Результат: HTML Help
- Имя: FAR

  Стоимость: $38 (условно-бесплатный продукт, полнофункциональный)

  Системные требования: Win 95

  Web-узел: `http://helpware.net/FAR/index.html`

  Результат: формат HTML Help, компилируемый из содержимого Web-узлов
- Имя: ForeHelp

  Стоимость: урезанная демонстрационная версия (максимум 20 тематических разделов) (~$100–$700)

  Системные требования: Win 95

  Web-узел: `http://www.forehelp.com/`

  Результат: HTML Help, WinHelp и пр.
- Имя: HELLLP!

  Стоимость: условно-бесплатный продукт ($30–$100)

  Системные требования: Win 95/NT 4, Word 97/2000

  Web-узел: `http://mindlink.net/Ed_Guy/helllp.html`

  Результат: Windows 95
- Имя: Help Authoring Expert

  Стоимость: условно-бесплатный продукт ($65). Незарегистрированная версия имеет ограничение по количеству тематических разделов.

  Системные требования: Win 95/NT 4, Word 95/97

  Web-узел: `http://www.stevesamuelson.com`

  Результат: Windows 95
- Имя: Help Express

  Стоимость: Shareware ($149). Незарегистрированная версия содержит информацию рекламного характера, которая встроена в каждый справочный файл.

  Системные требования: Win 95

  Web-узел: `http://www.chainware.com/docs/he3.html`

  Результат: Windows 95
- Имя: Help Maker Plus

  Стоимость: условно-бесплатный продукт ($45). Незарегистрированная версия ограничена малым количеством тематических разделов

  Системные требования: Win 3.x/95/2000, Word 6/95/97/2000

  Web-узел: `http://www.exhedra.com/exhedra/helpmakerplus/features.asp`

  Результат: Windows 3.x, Windows 95

- Имя: Help Pad

  Стоимость: условно-бесплатный продукт ($80)

  Системные требования: Win 95

  Web-узел: `http://www.gcnet.com/bw/hpad/`

  Результат: Windows 95, HTML Help
- Имя: HelpBreeze

  Стоимость: коммерческая демонстрационная версия. Ограничена малым количеством тематических разделов

  Системные требования: Win 95/NT, Word 6/95

  Web-узел: `http://www.solutionsoft.com/`

  Результат: Windows 95, HTML Help
- Имя: Helpburger

  Стоимость: условно-бесплатный продукт ($40). Незарегистрированная версия ограничена малым количеством тематических разделов

  Системные требования: Win 95

  Web-узел: `http://www.langdaledesigns.co.uk/hb/helpburger.htm`

  Результат: Windows 3.x, Windows 95
- Имя: HelpGen

  Стоимость: условно-бесплатный продукт ($30)

  Системные требования: Win 3.x/95/NT

  Web-узел: `http://www.rimrocksoftware.com/helpgen.html`

  Результат: Windows 95
- Имя: HelpHikes Pro

  Стоимость: условно-бесплатный продукт ($35)

  Системные требования: Win 3.x/95

  Web-узел: `http://web.superb.net/helphikes/`

  Результат: Windows 3.x, Windows 95
- Имя: HelpMe

  Стоимость: бесплатный продукт

  Системные требования: Win 95

  Web-узел: `http://home.wxs.nl/~verho037/jfprogramming.htm`

  Результат: Windows 3.x, Windows 95
- Имя: HTML Help Workshop

  Стоимость: бесплатный продукт

  Системные требования: Win 95/NT 4

  Web-узел: `http://www.microsoft.com/workshop/author/htmlhelp`

  Результат: HTML Help
- Имя: HyperText Studio

  Стоимость: коммерческая демонстрационная версия. Ограничена малым количеством тематических разделов.

Системные требования: Win 95

Web-узел: `http://webnz.com/olson/ProductsFrameset.htm`

Результат: Windows 3.x, Windows 95, HTML Help

- Имя: JavaHelp

  Стоимость: бесплатный продукт

  Системные требования: Win 95/NT, Solaris, MacOS 8.1

  Web-узел: `http://www.javasoft.com/products/javahelp/index.html`

  Результат: JavaHelp

- Имя: NetHelp2

  Стоимость: бесплатный продукт

  Системные требования: Win 95

  Web-узел: `http://home.netscape.com/eng/help/home/home.htm`

  Результат: NetHelp2

- Имя: RoboHELP

  Стоимость: 15-дневная коммерческая демонстрационная версия (~$2,000)

  Системные требования: Win 95/98

  Web-узел: `http://www.ehelp.com`

  Результат: Most help formats

- Имя: SOS Help! Info-Author

  Стоимость: 30-дневная коммерческая демонстрационная версия

  Системные требования: Win 95

  Web-узел: `http://www.lamaura.com/`

  Результат: Windows 3.x, Windows 95

- Имя: Visual Help Pro

  Стоимость: 30-дневная коммерческая демонстрационная версия

  Системные требования: Win 95

  Web-узел: `http://www.winwareinc.com/`

  Результат: Windows 3.1, Windows 95, HTML Help

- Имя: Windows Help Designer

  Стоимость: условно-бесплатный продукт ($49–$79)

  Системные требования: Win 95/NT

  Web-узел: `http://www.visagesoft.com/`

  Результат: Windows 95, HTML Help

Резюме

Ни одно приложение не должно быть передано конечному пользователю без справочной системы в какой-либо ее форме. Однако хуже отсутствия справки может быть только одно — плохая справка. Как показано в этой главе, справочную информацию можно "подавать" в виде различных "блюд", т.е. в разнообразных формах и форматах файлов. Форматом, который по-

прежнему остается самым популярным в мире Windows, является WinHelp, несмотря на то что Microsoft смещает акцент на более новый формат HTML. Для C++Builder формат WinHelp — это естественный выбор, поскольку он органично поддерживается компонентами библиотеки VCL. Поэтому C++Builder поставляется с компилятором Microsoft Help Workshop.

Однако самым трудным в создании справочной системы является не выбор "правильного" формата, а написание в лаконичной форме по-настоящему полезной информации. Очень важно тщательно спланировать справочные файлы и решить, какую информацию оформить в виде описания последовательности действий, а какую — в виде справочных, концептуальных или учебных разделов. На этом этапе также важно определить, какая информация действительно понадобится вашим пользователям. Поставьте себя на место конечного пользователя и постарайтесь почувствовать его потребности, только тогда вы сможете понять, как подготовить высококачественную электронную справочную систему.

Глава

28

Распространение программных продуктов

Саймон Ратли-Фрэйн
Филипп Х. Блантон II
Халид Алманай

В этой главе мы рассмотрим лучший способ распространения приложений и возможности извлечения максимальной пользы из Internet-маркетинга. Вы уже порядком потрудились, и нам хотелось бы облегчить вам жизнь, предложив несколько советов в плане того, как получить существенную отдачу от проделанной работы.

Языковая глобализация и локализация

Языковая глобализация — очень большая тема, заслуживающая отдельной книги. Создание приложения на различных языках — непростая задача, особенно, если речь идет о таких языках, как японский, китайский и арабский, в каждом из которых больше символов, чем можно представить с помощью 8-разрядного набора символов. Мы остановимся на локализации лишь для ASCII-ориентированных языков. Если имеется необходимость в локализации уникода (16-разрядного стандарта кодирования символов, позволяющего представлять алфавиты всех существующих в мире языков), вам придется обратиться к специальной литературе и проконсультироваться у специалистов в этой области. Однако, если вам достаточно локализации языков, которые представляются 8-разрядным кодом ASCII, эта глава вполне подойдет для создания по-настоящему локализованного приложения.

Общее представление о языковой глобализации

Существует несколько различных подходов к созданию многоязычного программного продукта средствами C++Builder. Один из них — использовать различные строковые файлы ресурсов для каждого языка, который вы собираетесь поддерживать, внести в исходный код соответствующую языковую версию и скомпилировать приложение для каждого языка. Недостаток такого подхода — увеличение стоимости поддержки, обусловленное необходимостью иметь различные версии продукта для каждого языка, что увеличивает сложность модернизации и внесения исправлений. Преимущество же в том, что есть возможность настроить приложение под каждый язык.

Другой метод (его мы здесь и рассмотрим) состоит в написании процедуры, которая будет находить и заменять заголовки и текстовые строки в приложении во время его выполнения, используя отдельный языковый файл ресурсов (`.ini`-файл) для каждого поддерживаемого языка. Это позволит конечному пользователю динамично изменять язык, выбирая нужный элемент из меню. Метод, который мы продемонстрируем, позволит вам добавлять поддержку нового языка в приложение, просто переведя `.ini`-файл на нужный язык, присвоив новому файлу соответствующее имя и разместив его в папке приложения.

Один из недостатков этого подхода — возможность (со стороны конечного пользователя) легко изменить текстовые файлы, в виде которых хранятся “языковые различия” приложения. Это можно предотвратить несколькими способами, например путем шифрования языковых файлов или хранения строк в небольшой таблице базы данных. Решение этой локальной проблемы мы оставляем вам. В большинстве случаев риск порчи текстовых файлов не оправдывается огромными усилиями, затраченными на их защиту. Здесь есть еще одна проблема — на чтение строк заголовков из `.ini`-файла может уходить много времени, если дело касается больших приложений. Поэтому в случае очень большого приложения `.ini`-файлы имеет смысл заменить таблицами базы данных. Безусловно, это внесет в приложение дополнительный уровень сложности, но зато существенно ускорит процесс загрузки нового языка.

Приложение Localize

На компакт-диске, прилагаемом к книге, находится исходный код приложения, именуемого Localize. Скопируйте содержимое папки с исходным кодом в подходящее место своего компьютера, скомпилируйте исходный код и выполните приложение Localize.exe. Вы должны увидеть на экране диалоговое окно, показанное на рис. 28.1.

Нетрудно заметить, что все заголовки написаны на английском языке. В начале работы приложение Localize использует строки, определенные в период разработки. Закрываемое приложение должно сохранять текущие установки языка в .ini-файле либо в системном реестре, а затем использовать их при следующем запуске.

Меню Language (Язык) в верхней части главной формы содержит элемент меню для каждого языкового файла ресурсов, имеющегося в папке приложения. В обработчике события Show главной формы MainForm мы вызываем метод FindLanguages(), который просматривает папку приложения, находя файлы, у которых имя совпадает с шаблонным именем *_Captions.con, а затем динамически создает новый элемент меню для каждого найденного файла языка. Эти .con-файлы имеют формат .ini-файла, а это значит, что мы можем использовать стандартные методы для чтения строк из файлов.

Чтобы перейти к новому языку во время работы приложения, пользователь просто выбирает из меню нужный язык. На рис. 28.2 показано приложение Localize после выбора французского языка (French) из меню Language.

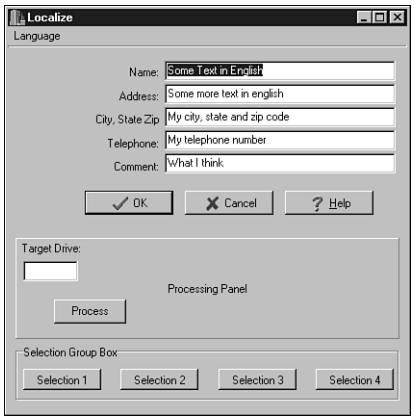


Рис. 28.1. Приложение Localize в начале работы

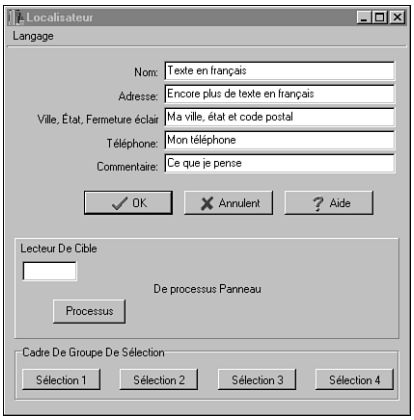


Рис. 28.2. Приложение Localize "разговаривает" на французском языке

Заглянув в папку приложения, вы найдете там четыре различных языковых файла, которые мы включили в данный пример.

```
English_Captions.con
French_Captions.con
German_Captions.con
Spanish_Captions.con
```

На заметку

Как видите, для этих файлов выбрано расширение .con, хотя в данном случае вы могли бы использовать и другое расширение. Мы остановили свой выбор на варианте .con, поскольку он не относится к зарегистрированным типам файлов, и поэтому операционная система не "будет знать", что с ним делать, когда ко-

нечный пользователь дважды щелкнет на его имени. Если бы мы использовали стандартное расширение .ini, то после двойного щелчка на этом файле Windows запустила бы редактор Notepad (Блокнот) и загрузила в него содержимое файла для редактирования.

Код приложения Localize

Код приложения Localize приведен в листинге 28.1. Загрузите его в свою среду разработки и опробуйте в работе, а затем вернитесь к этой главе для последующих разъяснений.

Листинг 28.1. Код реализации главной формы MainForm для программы Localize

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

#include "MainUnit.h"

#pragma package(smart_init)
#pragma resource "*.dfm"

TMainForm *MainForm;

void __fastcall LoadCaptions(TWinControl *Container, TIniFile *LangFile);

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

void __fastcall TMainForm::FormCreate(TObject *Sender)
{
     ComponentNames = new TStringList;
     Captions = new TStringList;
}

void __fastcall TMainForm::FormShow(TObject *Sender)
{
     FindLanguages();
}

void __fastcall TMainForm::OnLanguageSelection(TObject *Sender)
{
     Language = ((TMenuItem*)Sender)->Caption;
     RefreshCaptions();
}

void __fastcall TMainForm::FindLanguages (void)
{
     // Находим папку по умолчанию для языковых файлов и
     // создаем новый элемент меню для каждого найденного языка...
     TMenuItem *LangMenu, *NewLangSelection;
     TSearchRec sr;
```

```
     // На самом деле это излишне. Поскольку у нас есть только
     // один элемент меню верхнего уровня, мы могли бы просто
     // установить LangMenu = MainMenu1->Items->Items[0];
     LangMenu = MainMenu1->Items->Items[MainMenu1->Items->Count -1];
     if (FindFirst("*_Captions.con", 0, sr) == 0)
    {
        // Создаем новый элемент меню...
        NewLangSelection = new TMenuItem(LangMenu);
        // Устанавливаем заголовок равным первой части имени
        // языкового файла (эта часть заканчивается символом подчеркивания).
        NewLangSelection->Caption = sr.Name.SubString(1,
                                        sr.Name.Pos("_") - 1);
        // Устанавливаем обработчик события OnClick для нового
        // элемента меню равным OnLanguageSelection.
        NewLangSelection->OnClick = OnLanguageSelection;
        // Вставляем новый элемент в меню MainMenu
        LangMenu->Insert(LangMenu->Count, NewLangSelection);
        while (FindNext(sr) == 0)
        {
            // Выполняем тело цикла до тех пор, пока не исчерпаем
            // все языковые файлы.
            NewLangSelection = new TMenuItem(LangMenu);
            NewLangSelection->Caption = sr.Name.SubString(1,
                                        sr.Name.Pos("_") - 1);
            NewLangSelection->OnClick = OnLanguageSelection;
            LangMenu->Insert(LangMenu->Count, NewLangSelection);
        }
        FindClose(sr);
    }
}

void __fastcall TMainForm::RefreshCaptions(void)
{
    ComponentNames->Clear();
    Captions->Clear();
    TIniFile *NewCaptions = NULL;
    // Создаем имя файла для новых заголовков...
    String CaptionsFile = ExtractFilePath(Application->ExeName) +
                                    Language + "_Captions.con";
    // Проверяем, существует ли такой файл в папке приложения...
    // if (FileExists(CaptionsFile))
    {
        // Если файл существует, то создаем экземпляр
        // inifile-класса и открываем этот файл для чтения...
        try
        {
            NewCaptions = new TIniFile(CaptionsFile);
        }
        catch(...)
        {
            ShowMessage("Error opening the " + Language + " captions file!");
```

```
        }
    }
    else
    {
        // Если файл не существует, уведомляем об этом пользователя...
        MessageDlg("Could not find " + Language + "_Captions.con"
                + " !!!", mtError, TMsgDlgButtons() << mbOK, 0);
    }
    // Если NewCaptions имеет допустимое значение...
    if (NewCaptions)
    {
        // Получаем заголовки главной формы и меню...
        MainForm->Caption = NewCaptions->ReadString(
                        "MainForm", "FormCaption", "Localizer");
        LangMenu->Caption = NewCaptions->ReadString(
                        "MainForm", "LangMenu", "Language");
        // Опрашиваем все компоненты формы. Если они имеют
        // надписи в файле NewCaptions, загружаем соответствующие
        // значения из этого файла в заголовок компонента...
        LoadCaptions(MainForm, NewCaptions);
        delete NewCaptions;
    }
}

// Конец реализации формы MainForm.

void __fastcall LoadCaptions(TWinControl *Container,
                                        TIniFile *LangFile)
{
    for (int i =0; i < Container->ControlCount; i++)
    {
        if (dynamic_cast <TButton*>(Container->Controls[i]))
        { // Если компонентом является TButton...
            TButton& ref_obj = dynamic_cast<TButton&>(
                                    *Container->Controls[i]);
            String NewStr = LangFile->ReadString(
                            Container->Name, ref_obj.Name, "");
            if (strcmp(NewStr.c_str(), "")) ref_obj.Caption = NewStr;
        }
        if (dynamic_cast <TSpeedButton*>(Container->Controls[i]))
        { // Если компонентом является TSpeedButton...
            TSpeedButton& ref_obj = dynamic_cast<TSpeedButton&>(
                                      *Container->Controls[i]);
            String NewStr = LangFile->ReadString(
                             Container->Name, ref_obj.Name, "");
            if (strcmp(NewStr.c_str(), "")) ref_obj.Caption = NewStr;
        }
        else if (dynamic_cast <TLabel*>(Container->Controls[i]))
        { // Если компонентом является TLabel...
            TLabel& ref_obj = dynamic_cast<TLabel&>(
                                    *Container->Controls[i]);
```

```
        String NewStr = LangFile->ReadString(
                          Container->Name, ref_obj.Name, "");
        if (strcmp(NewStr.c_str(), "")) ref_obj.Caption = NewStr;
       }
       else if (dynamic_cast <TEdit*>(Container->Controls[i]))
       { // Если компонентом является TEdit, устанавливаем его
        // свойство Text...
        TEdit&ref_obj = dynamic_cast<TEdit&>(
                                  *Container->Controls[i]);
        String NewStr = LangFile->ReadString(
                         Container->Name, ref_obj.Name, "");
        if (strcmp(NewStr.c_str(), "")) ref_obj.Text = NewStr;
      }
      else if (dynamic_cast <TPanel*>(Container->Controls[i]))
      { // Если компонентом является TPanel, загружаем
       // заголовок панели (при наличии такового)...
        TPanel&ref_obj = dynamic_cast<TPanel&>(
                                   *Container->Controls[i]);
        String NewStr = LangFile->ReadString(Container->Name,
                                        ref_obj.Name, "");
        if (strcmp(NewStr.c_str(), "")) ref_obj.Caption = NewStr;
       // ...а затем рекурсивно вызываем функцию
       // LoadCaptions(), чтобы опросить все объекты
       // панели, и обновляем их заголовки или
       // текстовые строки...
        LoadCaptions(&ref_obj, LangFile);
      }
      else if (dynamic_cast <TGroupBox*>(
                                   Container->Controls[i]))
      { // Если компонентом является TGroupBox, загружаем
       // заголовок (при наличии такового)...
        TGroupBox&ref_obj = dynamic_cast<TGroupBox&>(
                                   *Container->Controls[i]);
        String NewStr = LangFile->ReadString(
                          Container->Name, ref_obj.Name, "");
        if (strcmp(NewStr.c_str(), "")) ref_obj.Caption = NewStr;
       // ...а затем рекурсивно вызываем функцию LoadCaptions(),
       // чтобы опросить все объекты, содержащиеся в GroupBox,
       // и обновляем их заголовки или текстовые строки...
       // LoadCaptions(&ref_obj, LangFile);
      }
    }
}
```

Как работает эта программа

Прежде всего необходимо отметить, что во время инициализации приложение `Localize` динамически создает меню Language в зависимости от “найденных” им языков. Самое замечательное здесь то, что перевод приложения на новый язык осуществляется очень легко. Достаточно сделать копию файла `English_Captions.con`, переименовать его так, чтобы в имени отражался новый язык, и позаботиться о его переводе на этот язык. После размещения языко-

вого файла в папке приложения и перезапуска этого приложения первая часть имени программы (до символа подчеркивания) появится в меню Language в качестве его нового элемента. Если пользователь выберет новый язык, приложение "само" выполнит перевод в соответствии с содержимым нового языкового файла.

Класс TMainMenu в C++Builder 5 приобрел (в сравнении с версией 4) много новых свойств. Одно из них — AutoHotKeys. Это свойство автоматически помещает в строку заголовка амперсанд (&), чтобы создать клавиши быстрого доступа к элементам меню. Это свойство по умолчанию устанавливается равным значению maAutomatic. В такой конфигурации имена файлов будут изменены, и приложение не сможет найти языковые файлы. Поэтому установите для свойства AutoHotKeys значение maManual либо устанавливайте его программным путем при создании новых элементов меню.

В качестве проверки удалите один из языковых файлов из папки приложения и перезапустите приложение. Вы заметите, что удаленного языка больше нет в меню Language.

При выборе элемента из меню Language строковая переменная Language устанавливается равной текущему выбору, после чего вызывается метод RefreshCaptions(). Этот метод создает новый объект TIniFile, открывает выбранный языковый файл, устанавливает заголовки формы и меню в соответствии со значениями в выбранном языковом файле, а затем вызывает метод LoadCaptions().

Метод LoadCaptions() в качестве параметров принимает указатель на контейнер компонента (в данном случае форму MainForm) и указатель на объект TIniFile. Затем выполняется опрос всех элементов управления в контейнере с целью поиска совпадающих строковых ресурсов в языковом файле. При отыскании элемента управления с совпадающей строкой в языковом файле эта строка загружается либо в свойство Caption, либо в свойство Text соответствующего элемента управления. Если элемент управления сам является контейнером, метод LoadCaptions() вызывается рекурсивно, чтобы загрузить все необходимые заголовки в элементы управления этого контейнера.

Стоит запомнить

Чтобы создать другие языковые файлы, используемые программой Localize, мы обратились по адресу: http://www.babelfish.altavista.com. Поскольку Web-узел BabelFish позволяет выполнить лишь дословный перевод, некоторые варианты словоупотребления иностранного языка выглядят топорно и даже неуместно. BabelFish можно использовать в качестве отправной точки (начинать ведь всегда трудно), но перед реальным "вводом в эксплуатацию" нового языка необходимо позаботиться о том, чтобы файлы ресурсов после перевода были просмотрены и отредактированы специалистами, в совершенстве владеющими этим языком. Конечно, если вы сами свободно разговариваете на разных иностранных языках, то просим извинить нас за излишние в этом случае предостережения. Читатель поймет наши опасения, поскольку иногда можно встретить довольно странные варианты перевода, как, например, в меню одного швейцарского ресторана: "Наши вина не оставляют надежды". (Вероятно, предполагалось следующее: "О наших винах не скажешь, что они оставляют желать лучшего".)

Мастер DLL-ресурсов

Как упоминалось в предыдущем разделе, существует много различных методов создания разноязычных программных продуктов. В C++Builder также предусмотрены средства для решения этой задачи. Такие средства, как мастер DLL-ресурсов (Resource DLL Wizard) и менеджер перевода (Translation Manager) — они входят только в версию Enterprise — настолько

упрощают процесс глобализации приложений, что вам не придется добавлять вручную ни единой строки кода.

Лучше всего для выполнения глобализации программного продукта поместить все интерфейсные строки проекта в отдельный файл. Этот файл затем переводится на нужные языки и распространяется вместе с приложением. Такой метод (с использованием `.con`-файлов) был продемонстрирован в предыдущих разделах. Именно здесь и уместно воспользоваться услугами мастера Resource DLL Wizard.

Как работает мастер

Мастер DLL-ресурсов несколько упрощает эту задачу путем создания библиотеки DLL для каждого выбранного вами языка. Эта DLL-библиотека содержит переведенные строки интерфейса проекта. Однако расширение у DLL-файла не `.dll`; вместо него используется комбинация из трех букв для задания языка и страны. Первые две буквы означают язык, а последняя — название страны. Например, расширением DLL-ресурса для канадского французского является `.frc`, где `fr` — это сокращение от слова *French* (французский), а `c` — первая буква слова *Canadian* (канадский).

Созданный файл DLL-ресурса обязательно нужно распространять вместе с приложением. Если основной параметр системы, "отвечающий" за местную специфику (т.е. национальную и культурную среду функционирования программы), совпадает с параметром, для которого был создан DLL-ресурс, то в приложении используется соответствующий язык. В противном случае приложение будет "говорить" на том языке, с помощью которого это приложение первоначально было создано. Например, если приложение было создано в системе, использующей американский (U.S.) английский, и при этом оснащено специально созданным DLL-ресурсом (`frc`) с ориентацией на канадский французский (Canadian French), то приложение будет работать с использованием этого `frc`-DLL-ресурса, если и система пользователя настроена на язык French (Canadian). В противном случае оно будет работать как English (U.S.)-приложение.

Как создать DLL-ресурс

Чтобы узнать, как это делается, давайте решим следующую проблему.

Допустим, вы создали приложение на английском (США) языке. Но вы хотели бы, чтобы оно поддерживало еще два языка, арабский (Бахрейн) и немецкий (Щвейцария). Как это сделать? Выполните следующие действия.

1. Откройте приложение. Если вы планируете перевести строки файла ресурсов проекта, выберите команду Project⇨Options⇨Packages (Проект⇨Параметры⇨Пакеты), отмените выбор опции Build with Runtime Packages (Создать проект с помощью пакетов времени выполнения), затем сохраните проект. Если это новый проект, то, прежде чем перейти к следующему этапу, завершите создание проекта и сохраните его.
2. В среде IDE выберите команду Project⇨Build Project (Проект⇨Построить проект).
3. Не закрывая проект, выберите команду File⇨New (Файл⇨Создать). Во вкладке New выберите вариант Resource DLL Wizard (Мастер DLL-ресурсов). Щелкните на кнопке OK, чтобы начать совместную работу с мастером.
4. Если вы не сохранили изменения в проекте, вам будет предложено их сохранить. Выберите вариант Yes.
5. Вы увидите вводный экран мастера DLL-ресурсов. Щелкните на кнопке Next (Далее).
6. Мастер отобразит экран с именем вашего проекта. Убедитесь, что выбран флажок, расположенный рядом с именем проекта. Щелкните на кнопке Next.

7. Мастер отобразит список всевозможных языков народов мира. В качестве проверки мы выберем из этого списка варианты Arabic (Bahrain) и German (Swiss) (с таким же успехом вы можете выбрать любые другие языки). Замечу, что в некоторых системах Windows опция German на этом и следующем этапах будет отображена в виде German (Switzerland). Щелкните на кнопке Next.
8. Откроется окно редактора пути. Здесь можно изменить предлагаемый по умолчанию адрес (путь), по которому будут скопированы файлы, предусмотренные для каждого языка. Чтобы изменить адрес "языковой" папки, выделите его, а затем введите новый адрес или щелкните на кнопке с многоточием, чтобы выбрать адрес путем просмотра каталогового дерева. Щелкните на кнопке Next.
9. Этап добавления файлов. Иногда возникает необходимость добавить другие файлы в проект, например файлы, предназначенные для перевода, но IDE об этом "ничего не знает". Кроме того, вы можете удалить файлы, добавленные ранее для перевода. Завершив перемещение файлов, щелкните на кнопке Next.
10. Откроется диалоговое окно редактора режима обновлений. В первом списке этого экрана будут отображены выбранные языки. Во втором столбце вы увидите режим обновления. Если файл DLL-ресурса был создан до обновления, вам будет предложен вариант Update (Обновить) или Overwrite (Перезаписать). Можно изменить режим обновления, выделив его и выбрав нужный вариант из комбинированного списка. Если для какого-либо языка DLL-ресурс не существует, в качестве режима обновления будет предложено Create New (Создать новый). Разобравшись с режимом обновления, щелкните на кнопке Next.
11. Теперь мастер оповестит вас о действиях, которые ему предстоит выполнить. Щелкните на кнопке Finish (Готово).
12. Если вы еще не компилировали проект, вам будет предложено это сделать, поэтому выберите вариант Yes. Если появится окно сообщений, уведомляющее о том, что опущен файл DCR (Delphi-compatible resource — совместимый с Delphi ресурс), выберите вариант Yes, чтобы создать его.
13. Откроется окно, в котором кратко сообщается о выполняемых действиях, а по завершении обработки появится диалоговое окно Processed (Обработка завершена). Здесь содержится вся статистическая информация о файлах, которые были созданы мастером DLL-ресурсов. Щелкните на кнопке OK.
14. На этом этапе мастер DLL-ресурсов завершил свою работу. В версии C++Builder 5 Enterprise будет вызван менеджер перевода (Translation Manager). Через мгновение мы вернемся к C++Builder.
15. Чтобы увидеть результаты работы мастера, запустите программу Windows Explorer и посмотрите на содержимое папки проекта. Вы увидите две созданных папки: `ARH` — для языка Arabic (Bahrain) и `DES` — для German (Swiss).
16. Просмотрев содержимое этих папок, вы увидите копию файлов своего проекта и RC-файл (Windows Resource, т.е. файл ресурсов Windows). Теперь необходимо перевести строки в файлах ресурсов. Закройте программу Windows Explorer и вернитесь к C++Builder. Если у вас нет версии C++Builder Enterprise, перейдите к п. 23; в противном случае продолжайте со следующего пункта (17).
17. В C++Builder 5 Enterprise был вызван Translation Manager. Вам предлагается отреагировать на следующее сообщение: `The Translation Manager requires all files to be saved. Save files and proceed?` (Менеджер перевода требует закрытия всех

файлов. Сохранить файлы и продолжить работу?). Выберите вариант Yes. Присвойте имя групповому файлу проекта и щелкните на кнопке Save (Сохранить).

18. Откроется диалоговое окно Translation Manager с двумя выбранными ранее языками, которые перечислены слева.
19. Для арабского языка, т.е. для варианта Arabic (Bahrain), расширьте элементы ресурсов (формы и сценарии ресурсов) в трех окнах слева.
20. Теперь нужно перевести строки для каждого элемента. Выделите первый из трех элементов и переведите все его строки, отображенные на правой панели (заголовки формы и надписи, имена дней недели и пр.). Повторите процесс перевода для каждого элемента ресурсов в трех окнах. Затем сохраните все изменения, щелкнув на пиктограмме сохранения или нажав клавиши <Ctrl+S>.
21. Повторите пп. 19 и 20 для языка German (Swiss).
22. Завершите работу с менеджером перевода, щелкнув на пиктограмме Exit (Выход) или нажав клавиши <Alt+X>.
23. Сохраните и закройте основной проект. Если у вас установлена версия C++Builder Enterprise, перейдите к п. 28.
24. Откройте файл проекта в папке `ARH`.
25. Теперь необходимо перевести на нужный язык интерфейсную часть проекта.
26. Вам также нужно перевести строки ресурсного файла проекта. Для этого выберите команду View⇨Project Manager (Вид⇨Менеджер проекта). Щелкните правой кнопкой мыши на RC-файле и выберите из контекстного меню команду Open (но не команду Open As Text). Теперь можно вручную перевести строки в столбце Text. Завершив перевод, сохраните и закройте проект. Помните, что если бы вы не сбросили опцию Build with Runtime Packages в п. 1, вы не увидели бы никаких строк для редактирования.
27. Повторите пп. 24–26 для проекта в папке `DES`.
28. После перевода для создания каждого DLL-ресурса необходимо создать проекты в "языковых" папках (папках `ARH` и `DES` в данном случае). По очереди откройте каждый проект, скомпилируйте его и закройте. Таким образом вы создадите все DLL-ресурсы.
29. Перейдите в корневую папку, где расположен ваш исходный проект. Там вы увидите два новых файла, `<ProjectName>.ARH` и `<ProjectName>.DES`, которые представляют DLL-ресурсы для языков Arabic (Bahrain) и German (Swiss), соответственно.

Как протестировать приложение

Чтобы протестировать созданные файлы, необходимо изменить языковые установки Windows. Если вы работаете в Windows 2000, выберите команду Settings⇨Control Panel (Настройка⇨Панель управления) из меню Windows Start (Пуск), а затем щелкните на пиктограмме Regional Options (Региональные параметры). Во вкладке General (Общие) выберите вариант Arabic (Bahrain) и щелкните на кнопке OK.

Если вы работаете в Windows 9x или NT, выберите из меню Windows Пуск команду Настройка⇨Панель управления, а затем щелкните на пиктограмме Regional Settings (Язык и стандарты). Во вкладке Regional Settings (Региональные стандарты) установите язык Arabic (Bahrain) и щелкните на кнопке OK. Если вам будет предложено перезагрузить систему, выберите вариант Yes.

Обратите внимание, что при инсталляции Windows могут быть доступны не все варианты языков, поэтому не исключено, что вы не сможете увидеть результат всех своих переводов.

Если окажется, что региональный стандарт Arabic (Bahrain) недоступен в вашей системе, перейдите к варианту использования немецкого языка.

Запустив приложение, вы увидите, что оно использует арабский интерфейс. Остановите работу приложения, измените региональный стандарт Windows, установив его равным German (Swiss), и снова запустите приложение. Теперь оно будет работать с использованием немецкого интерфейса. Опять остановите приложение и верните исходный региональный стандарт Windows, например русский.

На заметку Если вы работаете в Windows 9x, при изменении языка вам будет предложено перезагрузить систему. Мы рекомендуем так и сделать.

Если вы настроили свою систему на язык, отличный от упомянутых выше, ваш проект будет работать с использованием языка, установленного по умолчанию. Чтобы приложение "узнало" об изменении значения языкового параметра Windows и отобразило соответствующий интерфейс, необходимо перезапустить его.

Распространение других файлов и программ

При распространении приложения важно включить в поставляемый комплект все файлы, необходимые для его корректной работы на другом компьютере. Не стоит забывать, что конечный пользователь вашего приложения может находиться от вас "за тридевять земель", поэтому распространение неполного приложения не только крайне раздражает пользователя, но и чрезвычайно ухудшает ваши отношения с клиентами.

В мире вычислительной техники очень трудно заслужить доброе имя, но очень легко приобрести дурную славу. Поэтому при создании программы инсталляции со всей серьезностью отнеситесь к составлению списка дополнительных файлов. В этом разделе мы рассмотрим, какие еще файлы необходимо включить в приложение.

Файлы приложения

Существует ряд файлов, очень важных для конечных пользователей, но есть и такие, в которых пользователи не испытывают никакой потребности.

Ниже приведен список файлов, которые пользователи всегда стремятся найти.

- `ReadMe.txt`
- `help.hlp` (или новый эквивалент `*.chm`)
- `License.txt`

Теперь приведем список файлов, о существовании которых пользователь может не знать в силу уникальности их применения именно к данному приложению. При этом они могут быть очень важны для корректной работы вашего приложения.

- `Help.cnt`
- DLL-файл (файлы)
- файлы базы данных
- пакетные файлы
- другие файлы

Файлы из первого списка должны быть знакомы вам как программисту, поскольку, вероятно, вам не раз приходилось загружать или инсталлировать приложения, созданные другими компаниями.

Файл Readme.txt

Этот файл должен содержать важную информацию, к которой пользователь может легко получить доступ. Если вам нужен контактный адрес для приложения, то где вы будете искать его прежде всего? Конечно же, в файле `Readme.txt`. Этот файл должен включать общую информацию о приложении (о том, что оно делает, как его инсталлировать и как его удалить), а также данные о вашей компании (контактные телефоны, Web-адрес и другие приложения, созданные вашей компанией).

Важно скомпоновать свой файл `Readme.txt` таким образом, чтобы конечный пользователь мог легко получить доступ к необходимой ему информации. Не следует заставлять пользователя (когда у него возникнут к вам вопросы) просматривать подряд 20 страниц Readme-аннотации только для того, чтобы найти адрес вашей электронной почты. Перечислите основные элементы, содержащиеся в этом файле, и пронумеруйте их, и если адрес вашей электронной почты приведен в разделе 5, то пользователь сразу же обратится за ним именно к этому разделу.

Было бы очень разумно с вашей стороны перечислить все инсталлируемые приложением файлы с указанием их местоположения. Такого рода информация вселяет в пользователя больше уверенности при работе с приложением.

Файл Help.hlp или help.chm

Каждое приложение должно иметь справочный файл, связанный с приложением таким образом, что если пользователю при выполнении какой-либо задачи понадобится справка, он сможет легко ее получить. На добросовестное написание справочного файла не стоит жалеть времени и сил, поскольку он существенно сэкономит ваше драгоценное время после того, как приложение окажется в руках пользователей. В противном случае вам придется затратить уйму времени на информационную поддержку своих клиентов по электронной почте или по телефону. Лучше уделить больше внимания дальнейшему усовершенствованию своего приложения, чем тратить время на поддержку старой версии.

Справочный файл должен содержать ту же информацию, которую вы включили в файл `Readme.txt`, и многое другое. Сюда следует включить максимально возможный объем полезной для пользователя информации (данные о контактах, лицензионное соглашение, Web-адрес и пр.).

Файл License.txt

Этот файл очень важен, и каждое распространяемое вами приложение должно его иметь. Он содержит информацию о сроках и условиях использования приложения. В нем должен быть описан процесс получения разрешения на его дальнейшее распространение. Если приложение относится к разряду условно-бесплатных, вам, вероятно, хотелось бы, чтобы оно получило как можно более широкое распространение, но при этом вы могли бы указать, что оно должно распространяться только в форме программы инсталляции.

Кроме того, в этом файле может содержаться требование, чтобы любой издатель журнала или книги, прежде чем распространять ваше приложение в какой бы то ни было форме, должен непременно связаться с вами. Это позволит вам отслеживать географию использования вашего приложения и даст шанс на получение экземпляра такого журнала или книги.

Файл Help.cnt

В файле Help.cnt хранится содержание вашего справочного файла. Об этом файле часто забывают, поскольку справочный файл приложения действует и выглядит должным образом. Можно потратить не один час на увязывание всех частей справочного файла, но если вы используете надежный компилятор справочной системы, файл содержания создается автоматически.

Попробуйте удалить *.cnt-файл из справочной системы, а затем запустить ее, чтобы увидеть последствия таких действий. Вместо раздела содержания будет отрыта первая страница справочного файла. Если не увязать все составные части между собой, такой справочный файл может лишь ввести пользователя в заблуждение.

DLL-файл (файлы)

Если вы используете DLL-файлы, не забудьте включить их в общий список вместе с другими файлами, распространяемыми с приложением. В противном случае ваше приложение или не будет работать вообще, если DLL-файлы скомпонованы статически, или не будут работать некоторые функции, если имела место динамическая компоновка. В последнем случае ваше приложение будет работать до того момента, где должен загружаться отсутствующий DLL-файл.

Файлы базы данных

Если вы создали приложение, работающее с базой данных, вам придется присоединить к нему таблицы базы данных и другие связанные с базой данных файлы. Необходимо также убедиться в том, что приложению эти файлы "видны", т.е. доступны, и оно может использовать их после инсталляции на компьютере конечного пользователя.

Стоит также подумать о том, все ли конечные пользователи будут обеспечены необходимым ядром базы данных, чтобы успешно использовать файлы базы данных в вашем приложении. Нелишне также спросить себя, а сможет ли каждый пользователь установить эти файлы базы данных. Если у вас возникнут на этот счет сомнения, то лучше включить в программу инсталляции ядро базы данных Borland.

В главе 29 мы рассмотрим, как использовать программу InstallShield Express, которая поставляется вместе с C++Builder. Мы покажем, как использовать раздел базы данных программы InstallShield, чтобы не беспокоиться о том, оснащен ли компьютер пользователя ядром базы данных.

Если у вас есть база данных, то неплохо бы иметь несколько примеров того, как она выглядит при ее корректном использовании, или снабдить пользователя шаблонами, которым он мог бы следовать. Никогда не следует полагаться на опыт пользователя. Возможно, некоторые из них действительно будут иметь определенный опыт, но обязательно найдутся абсолютные новички в этой области. Вы же должны удовлетворить запросы пользователей всех категорий.

Пакетные файлы

Эти файлы используются приложением во время его выполнения и в период разработки. Они понадобятся конечному пользователю в том случае, если вы скомпилировали свое приложение при установленной опции Build with Runtime Packages (Строить с использованием пакетов времени выполнения). Эта опция устанавливается во вкладке Packages (Пакеты) диалогового окна Project Options (Параметры проекта).

Файлы, которые нужно присоединить к приложению, различны для разных приложений; тем не менее они должны быть включены в *.EXE-файл вашего приложения. В противном случае конечный пользователь не сможет выполнить приложение.

При использовании пакетов нужно найти компромиссное решение. С одной стороны, пакет уменьшает размер `*.EXE`-файла, но, с другой стороны, для обеспечения работоспособности приложения вам придется использовать большие `*.bpl`-файлы. Поэтому вся выгода от использования меньшего по размеру файла приложения теряется из-за необходимости включения в общий комплект поставки дополнительных файлов.

Другие файлы

Какие же еще файлы обычно используются разработчиком для обеспечения корректной работы приложения? Нетрудно забыть о WAV-файле, который вы воспроизводите, когда пользователь щелкает на кнопке, или текстовом файле, который загружается при запуске приложения в работу. Отсутствие одного из таких маленьких файлов может сделать работу всего приложения некорректной и привести к огромному потоку запросов по электронной почте на предмет технической поддержки.

Тестирование созданного приложения на другом компьютере — очень важный этап разработки, позволяющий понять, какие дополнительные файлы требуются вашему приложению, и что произойдет, если их не будет. Гораздо лучше сделать ошибку на своем компьютере, чем на компьютере конечного пользователя.

Этапы распространения приложения

Убедитесь в том, что вы точно знаете, какие еще нужны файлы, чтобы заставить приложение работать на другом компьютере. По возможности протестируйте свое приложение на чужом компьютере. Некоторые разработчики предпочитают, чтобы тестированием занимались другие, и стремятся проверить свое приложение на наличие ошибок и степень дружелюбия со стороны пользователей ценой представления им бесплатной копии приложения, которое они тестируют. Поищите в Internet информацию на тему “Beta Testers”.

Найдите желающих протестировать созданное вами приложение или его новую версию. Как правило, посторонние люди способны более объективно оценить ваше приложение и более честно его охарактеризовать, чем друзья и знакомые, которые, возможно, не решатся вас обидеть нелестным отзывом. Уделите достаточное внимание созданию полноценного и четкого справочного файла и не забудьте присоединить его к своей программе, чтобы ее можно было легко использовать.

Защита авторских прав и лицензирование программных продуктов

Создав приложение, следует позаботиться о его защите. Одна опасность состоит в угрозе кражи вашей работы. Другая — в возможности легальных действий против вас, если кто-то, используя ваше приложение, потеряет данные или пострадает другим образом.

Уведомление об авторском праве защищает ваше приложение от попытки продать его или использовать как свое собственное творение. Лицензия на использование пакета программ описывает ваши намерения относительно распространения приложения или его презентации в том случае, если кто-либо решит поместить его на страницы своего Web-сервера или компакт-диск. В лицензионном соглашении можно также оговорить, что вы не берете на себя ответственность за потерю данных.

Защита авторских прав

Ниже приведен список нескольких Web-узлов, которые позволят получить вам более подробную информацию по защите авторских прав.

- `http://www4.law.cornell.edu/uscode/17/`. Здесь можно получить информацию и текст законов по защите авторских прав в Соединенных Штатах.
- `http://www.ipaustralia.gov.au/`. Этот узел содержит информацию об интеллектуальной собственности в Австралии и включает гиперссылки на другие Web-узлы, в том числе и на узел Australian Copyright Council (Австралийский совет по защите авторских прав) по адресу: `http://www.copyright.org.au/`.
- `http://www.patent.gov.uk/`. Здесь можно найти информацию и текст законов по защите авторских прав в Соединенном Королевстве (Великобритании и Северной Ирландии).

Легко спутать авторские права с патентами. Достаточно четкие определения этих понятий можно найти на Web-узле `ipaustralia` (его адрес указан в приведенном выше списке).

Уведомление об авторских правах состоит из четырех элементов: имя вашего приложения, слово *copyright* или символ `@`, год выпуска приложения и ваше имя или название компании. Включив эту информацию в свое приложение, вы выполняете минимальное условие для законной защиты своих авторских прав.

Приведем примеры уведомлений об авторских правах.

- My Application @ 2000, John Doe
- My Application Copyright 2000, John Doe

Авторские права необязательно регистрировать в специальном комитете. Если вы все-таки хотите это сделать, напишите в местный комитет по защите авторских прав (его адрес вы найдете в телефонном справочнике или с помощью Internet). Сотрудники этого комитета должны обеспечить вас бланком заявления. Несмотря на существование международных правил защиты авторских прав, в каждой стране эти правила и положения могут слегка отличаться. Регистрация авторских прав выполняется не бесплатно, и размер взноса в разных странах различен.

Куда поместить уведомление об авторских правах

Где разместить уведомление об авторских правах — решать вам, но рекомендуется отобразить его в окне About (О программе) вашего приложения. Используйте средство C++Builder Version, чтобы уведомление об авторских правах встроить в `.EXE`- или `.DLL`-файл вашего приложения. Результат можно затем увидеть в программе Windows Explorer, выделив нужный `.EXE`- или `.DLL`-файл и проверив его свойства. Кроме того, это уведомление стоит разместить во всей сопровождающей документации (справочные файлы, Readme-файлы и на страницах Web-узла).

Полезные советы по защите авторских прав

Узнайте все, что можно, о законах, действующих в вашей стране, в отношении авторских прав Не забудьте включить соответствующие уведомления об авторских правах в свое приложение и во все сопровождающие его документы, включая справочные файлы. Лицензии на программные продукты должны быть доступны и распространяться с каждым разработанным вами приложением. При этом все равно стоит внимательно просматривать содержимое Web-страниц, чтобы не столкнуться позже с большими неприятностями. Используйте для этого приведенный выше список Web-адресов и выполните свою часть работы.

Лицензирование программного продукта

Это легальная информация, которую желательно знать вашим конечным пользователям. Вы найдете на своем жестком диске множество лицензий на программные продукты, например ту, что используется компанией Borland/Inprise (откройте для этого корневую папку C++Builder 5).

Большинство лицензий на программные продукты сходны между собой, и их можно разделить на следующие разделы.

1. Имя приложения и название компании.
2. Условия лицензии (например: "Устанавливая, копируя, загружая, осуществляя доступ или иным образом используя указанное программное обеспечение, Вы тем самым принимаете на себя условия настоящего соглашения. Если Вы не согласны с условиями настоящего лицензионного соглашения, не устанавливайте и не используйте данное программное обеспечение…").
3. Тип гарантийных обязательств, если таковые предусмотрены.
4. Любая информация о приложении, которую, по вашему мнению, необходимо знать конечным пользователям.
5. Форма правовой защиты на случай потери или повреждения компьютера конечного пользователя. Лицензия программного продукта должна содержать примерно следующее:

   "Ни при каких обстоятельствах *ваша компания* не обязуется возмещать ущерб вследствие каких-либо разрушений (включая потерю прибыли, перерывы в работе, утерю деловой информации и пр.), возникших в результате использования или невозможности использования этого продукта, даже если *ваша компания* была осведомлена о возможности таких разрушений".

Безусловно, прежде чем создавать и публиковать лицензию на использование своего программного продукта, следует получить квалифицированный совет юриста.

Это не окончательный список, но он создает прочный фундамент, на котором может быть построена лицензия, содержащая достаточно информации для самозащиты на случай критических обстоятельств.

Защита программных продуктов

Если вы отдали многие часы своему приложению и хотите выпустить его для широкого, но не бесплатного использования, необходимо позаботиться о защите своего детища, обеспечив так называемый пробный вариант продукта ("try before you buy" — перед тем как купить, попробуй).

Существует множество способов защитить свое приложение — от использования компонентов сторонних производителей до внесения дополнительного кода в само приложение, причем все варианты легко реализуемы, и при правильном выборе набора компонентов на это может уйти не более часа.

Ниже в этой главе мы поговорим о двух наборах компонентов от компании AHM Triton Tools (`http://www.tritontools.com`) и компании MJ Freelancing (`http://www.mjfreelancing.com`). Оба набора распространяются на условно-бесплатной основе и представляют собой высококачественные компоненты, которые отличаются простотой применения и высокой надежностью. В Internet можно найти и другие наборы компонентов, выполняющие аналогичные функции. Некоторые адреса приведены в конце этого раздела.

Защита приложений

С помощью нескольких строк кода, добавленных в условно-бесплатное приложение, можно заставить конечного пользователя зарегистрироваться или заплатить за продукт, если он хочет продолжать его использование. Воспользовавшись следующим кодом, вы защитите свое приложение ограничением Date Disable, обеспечивающим неработоспособность продукта после наступления заданной даты.

```
void __fastcall TForm1::FormShow(TObject *Sender)
{
     AnsiString TodaysDate;
     TodaysDate = DateToStr(Date());
     if (TodaysDate >= "21/10/00") // Для США формат даты другой: MM/DD/YY
     {
          ShowMessage("Date Reached"); // Дата наступила
          Close();
     }
}
```

Здесь всего лишь сравнивается текущая дата с датой запрещения работы приложения. Если оно будет открыто в этот день или позже, для пользователя будет отображено сообщение, а программа закрыта для дальнейшего использования.

Защита приложения с помощью компонентов сторонних производителей

На моем Web-узле представлено несколько условно-бесплатных программных продуктов, и в большинстве из них используются два компонента безопасности от компании MJ Freelancing: AppLock и ShareReg.

Компонент AppLock используется главным образом для блокировки приложения до тех пор, пока пользователь не введет код “отмычки”, который вы сгенерируете. При этом не требуется создавать никаких заплат или новых файлов и отправлять их пользователю, поскольку этот компонент использует защитный экран, в котором принимается информация о регистрации, необходимая для того, чтобы разблокировать приложение.

ShareReg — это стандартный компонент, предоставляющий разработчику всю необходимую для приложения защиту с использованием знакомого пользователю формата. Он позволяет установить либо защиту Date Disable, либо блокировку Runtime/Days Lock. Оба метода подробно описаны ниже в этой главе.

Вам предстоит сделать выбор из трех вариантов защиты: Days Run, Date Disable или комбинации первых двух. Этот компонент позволяет установить защиту, связанную с идентификационным номером (ID) компьютера, которая означает, что конечный пользователь не сможет использовать тот же код “отмычки” на другом компьютере, поскольку в защите используется серийный номер, считываемый с жесткого диска пользователя.

Ниже приводится фрагмент текста из документации на компонент ShareReg.

> Компонент `TShareReg` предназначен для таких условно-бесплатных приложений, копии которых, по вашему убеждению, будут передаваться другим потенциальным пользователям, НО при этом вы будете защищены знанием того, что зарегистрированные пользователи не смогут передать детали своей регистрации для предоставления “зарегистрированных” версий своим друзьям. Уровень безопасности определяется пользователем компонента (т.е. разработчиком). Конечно, совсем нетрудно передать

другим имя пользователя и пароль, но простым изменением одного или нескольких свойств можно ограничить работу приложения одним компьютером и пользователем.

На базе этих компонентов пишется исходный код для создания вашего собственного генератора разблокирующего кода, который вы можете включить в свою программу, например в базу данных пользователя. Разблокирующие коды генерируются из трех (максимум) полей, показанных на рис. 28.3: User Name (Имя пользователя), Organization (Организация) и Unlock Code (Разблокирующий код).

Рис. 28.3. Компонент ShareReg в действии. Отображает три поля, используемые для генерации разблокирующего кода

Одному приложению может быть назначен случайный разблокирующий код как минимум из 5,0E494 комбинаций. Это очень хорошая защита против хакеров.

Другая особенность этого компонента состоит в том, что информация о регистрации хранится в системном реестре компьютера конечного пользователя, причем ее можно записать в зашифрованном виде. Поэтому хакер не сможет отредактировать реестр, установить дату, выполнить подсчет, и пр. Если конечный пользователь изменит в своем компьютере дату, чтобы продлить срок использования приложения, он увидит сообщение о том, что это против правил эксплуатации условно-бесплатного приложения, после чего приложение будет закрыто.

Компонент ShareReg дает возможность создать частично заблокированную версию вашей программы, позволяющую конечным пользователям лишь опробовать ее. Только после того как ваше приложение будет успешно зарегистрировано конечным пользователем, блокировка будет снята. Возможно, вам приходилось видеть, как в условно-бесплатных программах до регистрации программы реализуется запрещение функции сохранения (Save). Этот компонент предоставляет и другие возможности. Например, вы можете добавить собственное растровое изображение для экрана заставки, разместить текст лицензионного соглашения на этом экране, чтобы пользователи могли с ним ознакомиться, и многое другое. Более подробную информацию по установке этих компонентов можно найти на страницах Web-узла компании MJ Freelancing по адресу: `http://www.mjfreelancing.com`.

Компонент AHMAppManager

AHMAppManager — еще один простой в применении компонент из набора AHM System. С его помощью довольно легко превратить совершенно незащищенное приложение в условно-бесплатное. Он позволяет установить функцию Date Disable, благодаря которой ваше приложение становится неработоспособным после достижения установленной даты. Здесь не предусмотрен ввод разблокирующего кода, поэтому для зарегистрированных пользователей вам придется сформировать отдельные приложения или заплаты.

Защита приложения с помощью других типов компонентов

В Internet можно найти множество различных наборов компонентов или DLL-библиотек, предназначенных для защиты приложений в процессе их условно-бесплатного распространения.

Компонент AppReg

`http://www.thebits.org`

Бесплатный компонент, который был создан интерактивными "слушателями" одного из обучающих разделов Web-узла TheBits. Чтобы найти этот компонент, нужно перейти в раздел компонентов этого узла.

Компонент RedRegistration

`http://www.redware.net/RedRegistration/`

Чтобы добиться достаточной степени безопасности, этот компонент использует шифрование высокого уровня. RedRegistration работает с любыми средами разработки, которые поддерживают DLL-библиотеки или элементы управления ActiveX. При этом нет необходимости использовать библиотеки динамической компоновки. Кроме того, пользователи могут приобрести здесь RedRegistration-блоки статических библиотек C++.

Компонент ShareLock

`http://www.nesbitt.com/ShareLock/index.html`

Простой в применении компонент, способный с помощью одной строки кода превратить любое 32-разрядное Windows-приложение в пробную версию. Компонент ShareLock 2.0 включает VCL, ActiveX, и DLL-версии для использования с любым языком программирования Windows.

Комплект Crypto++

`http://www.sampson-multimedia.com/cpp/`

Crypto++ SDK — комплект инструментальных средств, предназначенный для защиты и лицензирования программных продуктов с шаблонами для C++Builder. Он включает возможности передачи лицензии, интерфейс с системами отслеживания клиентов, автоматическое санкционирование посредством Web и многое другое.

Некоторые замечания по защите программных продуктов

Превратить бесплатное приложение в условно-бесплатное проще, чем могло показаться вначале. С помощью нескольких строк кода (затратив несколько минут) можно заложить в приложение "бомбу", которая выведет его из строя в нужный день. Если вы хотите создать более профессиональное приложение, стоит воспользоваться адресами компонентов, приведенными выше.

Условно-бесплатный способ распространения программных продуктов — прекрасный метод познакомить компьютерных пользователей со своим приложением. Чем больше пользователей запустит вашу программу на своем компьютере, тем больше вероятность продать ее.

Условно-бесплатные приложения

Приняв решение о том, чтобы сделать свое приложение условно-бесплатным, нужно выбрать оптимальный метод защиты. Этот раздел и посвящен описанию преимуществ и недостатков различных методов защиты.

Все больше приложений становятся условно-бесплатными. Вы днями и ночами разрабатывали приложение, отлаживали его и тестировали, поэтому вполне заслуживаете вознаграждения за свой труд. Если ваше условно-бесплатное приложение будет пользоваться успехом, значит, вы сможете позволить себе больше времени уделить его усовершенствованию и разработке нового, что в конечном счете выгодно всем пользователям.

Несомненно, создание условно-бесплатных приложений выгодно в финансовом отношении их создателям, однако не стоит слишком легко относиться к принятию решения идти именно таким путем. Ведь существуют определенные аспекты, которые нужно учитывать, прежде чем выпускать в свет свое творение.

Прежде всего следует подумать о поддержке своих пользователей. Кто будет отвечать на их бесконечные вопросы? Нужно ли вам выделять специальный адрес электронной почты для реализации поддержки и получения отзывов от пользователей? Нужно ли иметь раздел FAQ (часто задаваемых вопросов) на Web-узле приложения? Какого типа Web-страницы потребуются вашему приложению? Каким образом будет осуществляться покупка вашего приложения? Кто будет отвечать за финансовые вопросы? Можете ли вы согласиться на реализацию продаж посредством кредитных карточек? Все эти вопросы обязательно "свалятся на голову" удачливому автору условно-бесплатного приложения.

Теме распространения приложений через Internet и вопросам маркетинга мы уделим внимание далее в этой главе. Мы также поговорим о том, какого типа Web-страницы вам следует поддерживать и как лучше организовать продажу по кредитным карточкам, не создавая защищенные Web-страницы на своем узле.

Защита условно-бесплатных приложений

Как уже было сказано, условно-бесплатное приложение означает "опробуй прежде чем купить". Это прекрасный (если не идеальный) способ соблазнить компьютерных пользователей всего мира на то, чтобы они бесплатно опробовали вашу программу, но при этом не предоставлять им полную рабочую версию на неограниченный срок.

Основные способы защиты приложения от постоянного использования следующие:

- Date Disable;
- Runtime Lock;
- Days Lock;
- Disabled Functions.

Метод Date Disable

При использовании этого метода ваше приложение получит дату, по достижении которой приложение больше работать не будет или будут недоступны некоторые его функции. После того как пользователь заплатит за полную рабочую программу, вы сможете предоставить ему заплату или разблокирующий код (все зависит от того, как вы реализуете функцию Date Disable).

Эти методы защиты просты для реализации, но требуют определенной работы и создания дополнительных файлов проекта, поскольку вы должны иметь возможность (с помощью дополнительных файлов или заплат) избавиться от функции Date Disable. Основное преимущество этого метода — простота реализации и управления, а также надежность. Недостатком является необходимость постоянной перекомпиляции приложения с новой датой и последующей его передачи по назначению. После покупки приложения вам придется отправлять заплаты или полные файлы, что зависит от конкретной реализации защиты вашего условно-бесплатного варианта приложения.

Когда сеть Internet стала очень могучим инструментом маркетинга и многие владельцы Web-узлов с условно-бесплатными программами предпочитают хранить ваши файлы на своих серверах, а не компоновать их с главным файлом на вашем Web-узле, этот метод защиты условно-бесплатных программ стал неприемлем. Этот метод может даже тормозить развитие отношений между производителем и покупателем, поскольку многие Web-узлы имеют собственные компакт-диски, которые они передают своим клиентам. Дело в том, что на создание компакт-диска для передачи пользователю могут уйти недели приготовлений, а за это время ваш защищенный файл может устареть.

Метод Runtime Lock

В основу этого метода положена способность приложения определять количество его запусков. Самый простой способ узнать, сколько раз запускалось приложение, — сохранять соответствующие значения в реестре и считывать их.

Сделайте элементы реестра скрытыми, чтобы хакерам было очень трудно отыскать их и изменить. Если это не станет преградой для особо настырного хакера, то по крайней мере охладит пыл многих других.

Преимущества этого метода в том, что вы можете хранить файлы своего приложения на Web-сервере условно-бесплатных программ и распространять их посредством компакт-дисков, поскольку они не содержат заранее установленных дат, о которых стоит беспокоиться. Этот метод не требует повторных компиляций, поскольку не нужно устанавливать никаких новых переменных.

Недостаток метода состоит в том, что конечный пользователь теоретически сможет запускать приложение только в течение нескольких дней. Это может сильно повредить вашим намерениям продать свою программу; ведь вы хотели бы, чтобы пользователь всесторонне оценил результаты вашего труда и, более того, чтобы ваше приложение настолько ему понравилось, что он захотел бы его купить.

Метод Day Lock

Этот метод определяет, в течение какого периода будет работать приложение. Обычно период Day Lock составляет 30 дней. Для хранения даты инсталляции (чтобы определить количество дней эксплуатации) используется системный реестр. По исчерпании установленного периода должна сработать защита. Тип защиты выберите по своему усмотрению: это может быть запрещение отдельных функций или простое отображение сообщения для пользователя с последующим завершением работы программы.

А теперь о слабых сторонах этого метода. При покупке приложения вам нужно отослать покупателю заплаты или полные файлы. Это зависит от конкретной реализации защиты условно-бесплатного приложения.

Хранение приложения на чужом Web-сервере не является проблемой, поскольку файл не будет работать после определенной даты. Конечный пользователь сможет использовать его как посчитает нужным в течение установленного заранее периода времени, по истечении которого приложение не будет работать совсем либо частично (это также зависит от выбранного вами решения).

Метод Disabled Functions

Этот метод, вероятно, является самым простым для реализации. Поэтому на рынке найдется довольно много условно-бесплатных приложений, в которых используется этот метод (например, программа редактирования музыки Cool Edit). При использовании этого метода

вы передаете своим пользователям полную версию своего приложения, но с несколькими недоступными функциями, например Save (Сохранить) или Open (Открыть).

Реализация этого метода совершенно проста. В C++Builder установите для свойства `enable` выбранной функции значение `false`. Этого достаточно, чтобы превратить ваше приложение в полностью защищенное условно-бесплатное приложение.

Преимущества этого метода — простота реализации, полный контроль над количеством запрещаемых функций и возможность для пользователя работать с вашим продуктом в пробном режиме неограниченное время. При этом вы сможете хранить свое приложение на Web-сервере условно-бесплатных программ и компакт-диске, поскольку оно не содержит никаких заранее устанавливаемых переменных.

Без недостатков не обошлось и в этом случае — вам придется отсылать зарегистрированным пользователям заплату или полный файл.

Реализация методов защиты условно-бесплатных приложений

Существует множество способов реализации защиты условно-бесплатных приложений: от нескольких строк кода до использования компонентов сторонних производителей. Эти методы защиты подробно описаны в разделе "Защита программных продуктов" этой главы.

Методы защиты условно-бесплатных приложений

Не существует единого способа для создания условно-бесплатных приложений. Различные приложения требуют применения различных методов.

Выбор метода зависит от типа разрабатываемого приложения и ваших личных потребностей. Мой опыт показал, что для приложений общего назначения лучше всего подходит метод Day Lock. Дело в том, что этот метод защиты широко используется в компьютерном мире, и пользователи уже хорошо знакомы с ним и четко представляют себе, что произойдет по истечении заданного периода времени.

При этом важно заранее уведомить пользователя (включив в комплект поставки соответствующие документы) об условиях использования вашего приложения, т.е. о том, что оно является условно-бесплатным, о точном периоде его апробации, по завершении которого приложение станет неработоспособным, а также о том, какие функции являются недоступными.

Распространение продуктов и маркетинг посредством Internet

В Internet всем хватает места — и большим компаниям, и малым. Сегодня трудно найти даже небольшую фирму, которая бы не имела своей Web-странички.

Миллионы людей время от времени "с головой" погружаются в океан сети Internet, поэтому она — прекрасное средство рекламирования программных продуктов и ознакомления с ними миллионов пользователей. Как "достучаться" до всех этих пользователей и как убедить их в необходимости загрузить и опробовать наш условно-бесплатный продукт — вот в чем вопрос.

Web-узлы

Если у вас еще нет Web-узла для поддержки созданных приложений, позаботьтесь о таковом. Существуют сотни провайдеров услуг Internet (Internet service providers — ISP), предлагающих свободный доступ к Internet. По вполне приемлемым ценам можно получить имя домена и вскорости стать обладателем Web-узла.

Web-узел должен иметь хороший дизайн с простой компоновкой, четко изложенной информацией о приложении (приложениях) и не слишком большим количеством графических изображений, которые значительно увеличивают время загрузки. Хорошо бы иметь при этом 56-килобайтный модем и компьютер высокого класса, но многие компьютерные пользователи все еще работают с компьютерами вчерашнего дня, оснащенными 28-килобайтными модемами.

Следует также подумать о специализированных Web-страницах, предназначенных для загрузки приложений. Многие компьютерные пользователи находят на основной индексной странице слово *Download* (Загрузить) и переходят именно на страницу загрузки.

Поддержка пользователей

Известно, что люди очень разные. Не менее разными являются и пользователи компьютерных программ. Многие потенциальные покупатели могут по электронной почте задавать вопросы только для того, чтобы узнать, насколько быстро и внимательно вы отреагируете на их послания. Вам следует установить для себя четкие принципы делового общения и по возможности обзавестись отдельным адресом электронной почты для своих приложений (или по крайней мере для вопросов по части технической поддержки).

Организуя техническую поддержку по электронной почте, необходимо помнить о следующем.

- На все вопросы по электронной почте нужно отвечать как можно быстрее, как правило, в течение 24 часов.
- Ответить нужно на каждый вопрос.
- Отвечать на вопросы следует в вежливой форме, независимо от степени глупости (по вашему мнению) или грубости самого вопроса.
- Предложите автору вопроса уведомить вас, если его удовлетворил ваш ответ или совет, который вы дали, привел к успешному решению проблемы.
- Не забудьте проинформировать своих адресатов о том, что они могут в любое время посылать вам по электронной почте сообщения с другими вопросами.

Общеизвестно, что на приобретение хорошей деловой репутации уходят годы, а на то, чтобы ее погубить, достаточно нескольких дней.

Рекламирование приложений

Существует множество возможностей для рекламы приложений, и, как правило, за рекламу нужно платить. Для Internet-рекламы в основном используются специальные заголовки, или *баннеры*, после щелчка на которых выполняется переход к Web-узлу, содержащему все, что может интересовать потенциального покупателя. Баннеры используются на любом Web-узле, который обслуживается “баннерной” компанией. Узлы, которые обслуживаются подобными компаниями, должны отвечать определенным критериям (минимальное количество посетителей в неделю, отсутствие непристойного содержимого и пр.).

Обычно предлагается четыре метода организации рекламы: плата за щелчок, плата за просмотр, плата по общему итогу и плата за покупку.

Плата за щелчок означает, что вы платите за каждое из рекламных объявлений, на котором щелкнул потенциальный покупатель. Это довольно неплохой метод, поскольку ваш Web-узел посещают те, кого заинтересовала информация, что уже является залогом успеха. Главное — на вашем Web-узле появились посетители, а дальше уже ваша задача заинтересовать их в покупке или пробном использовании приложения.

Плата за просмотр означает, что вы платите за каждый просмотр клиентом вашего рекламного объявления. Это один из самых неэкономных способов рекламирования приложения, поскольку вы платите только за то, что кто-то увидел ваш объявление, и нет никакой гарантии, что он ее прочитает, а тем более посетит ваш Web-узел. Поскольку вы платите за каждый раз, когда потенциальному покупателю демонстрируется ваше рекламное объявление, вам приходится оплачивать щелчки на страницах. Если пользователь использует какую-то Web-страницу только для того, чтобы щелкнуть на следующей странице, он может даже не дожидаться полной ее загрузки, чтобы перейти дальше, но вы все равно уже обязаны заплатить. Большинство рекламных объявлений располагаются в верху Web-страницы и загружаются первыми. Этот метод приносит чуть ли не самые большие доходы для владельцев Web-узлов.

Плата по общему итогу используется в том случае, когда вы устанавливаете общее количество демонстрируемых рекламных объявлений. Подобный вид оплаты обычно применяется в рекламных кампаниях, когда вы платите твердую цену за объявления, показываемые конкретное количество раз за определенную сумму. Но и в этом случае оплата производится за постраничные щелчки.

“Комиссионное вознаграждение” — самый простой для разъяснения метод платы за рекламу. Вы платите конкретную сумму каждый раз, когда покупатель, увидев рекламное объявление, посетит ваш Web-узел и купит приложение. Эта сумма различна и, как правило, согласовывается с компанией, занимающейся рекламой.

Бесплатные баннеры

Можно публиковать рекламные баннеры и бесплатно. Для этого на Web-узел Linkexchange (`//www.linkexchange.com/`) вы сообщаете, сколько Web-страниц содержит ваш Web-узел и на какое количество из них вы бы хотели поместить “взаимные” рекламные объявления. Затем вы разрабатываете баннер для своего приложения в соответствии с определенной спецификацией и предлагаете его для узла Linkexchange.

Представители Linkexchange проверяют ваш Web-узел — они должны удостовериться, что заявленное число страниц соответствует действительному. Затем они посылают вам HTML-код своего баннера для его вставки в ваши Web-страницы. Таким образом, ваш Web-узел используется для рекламы других приложений, а ваши рекламные объявления демонстрируются на чужих Web-узлах. Получается своего рода взаимовыгодное рекламное сотрудничество под девизом “ты — мне, я —тебе”.

Узлы, предназначенные для загрузки программных продуктов

Хороший способ заставить потенциального пользователя испытать ваше приложение — использовать столько “загрузочных” Web-узлов, сколько сможете найти. Вполне оправдали себя такие Web-узлы, как ZDNet (`//www.zdnet.com`), WinFiles.com (`//www.winfiles.com`) и Shareware.com (`//www.shareware.com`). Тысячи потенциальных покупателей каждый день используют эти узлы для загрузки приложений. Этот список далеко не исчерпывающий — существуют сотни подобных узлов.

ZDNet, один из самых больших узлов загрузки, предлагает загрузить файл вашего приложения на свой Web-сервер. Там оно подвергается независимому рецензированию и оценивается по "звездной" системе, в которой пять звездочек означают высшую оценку.

Другие Web-узлы загрузки просто вносят приложение в список и создают ссылку на основной файл приложения на вашем сервере. Загрузка приложения на все эти Web-узлы требует немалого времени, но осуществляется в основном бесплатно, хотя есть и такие узлы загрузки программных продуктов, за услуги которых необходимо платить. Мы советуем не иметь с ними дела и поддерживать отношения с бесплатными, тем более, что в последних недостатка не ощущается.

Чтобы занести приложение в списки "загрузочных" Web-узлов, вам придется заполнить и представить на рассмотрение форму, содержащую имя вашего приложения, описание и прямые связи загрузки.

После того как ваше приложение будет принято на узел загрузки, не исключено, что редакторы журналов и представители издательств книг начнут с вами переговоры о разрешении включить в свои публикации аннотацию на вашу программу. Мы не можем гарантировать, что такая судьба обязательно ожидает ваше приложение, но если оно приобретет достаточную популярность, эти узлы будут благоприятствовать вашему становлению и достижению будущих успехов в разработке программных продуктов.

Прием кредитных карточек и предоставление разблокирующих кодов

Следующий этап после принятия решения об Internet-рекламе — как получить деньги покупателя. Конечно, об этом стоит подумать еще в период разработки приложения. Самый распространенный метод получения денег от покупателя с последующим предоставлением ему полной программы — покупка у вас разблокирующего кода или кода регистрации. Затем покупатель вставляет этот код для разблокирования программы и получает полный доступ ко всем ее функциям.

Использование кредитных карточек для покупки программных продуктов

Кредитные карточки — самый быстрый способ перевода денег с одного банковского счета на другой. Для банков обслуживание по кредитным карточкам — обычная практика, но вам придется платить комиссионные отчисления за каждую проводимую операцию. При этом вам необходимо позаботиться о безопасности вашего Web-узла или страницы регистрации. Безопасность в Internet — весьма животрепещущая тема, и многие до сих пор отказываются использовать свои кредитные карточки в Internet из страха быть обманутым.

Другой вариант — использовать созданные сторонними производителями службы, работающие с кредитными карточками. Таких предложений довольно много, и большинство подобных служб имеют хорошую репутацию. Продумайте в деталях, что именно вы хотели бы получить, а затем с помощью Internet найдите наиболее подходящий для вас вариант.

Поинтересуйтесь у своих друзей и знакомых, которые уже продавали свои условно-бесплатные продукты, какими службами кредитных карточек они пользовались.

Советы по выбору служб кредитных карточек

Выбрав службу, свяжитесь с ней по электронной почте и задайте вопросы о предоставляемых ею услугах. Наведите справки о ее работе.

Если в течение некоторого разумного интервала времени вы не получите от этой службы ответа, забудьте о ней. Не стоит подвергать риску свои так тяжело заработанные деньги.

Различные службы кредитных карточек назначают различные ставки комиссионного вознаграждения. Обычно чем выше вы оцениваете стоимость своего программного продукта, тем выше ставка.

Какова частота выплат? В какой валюте будут осуществляться выплаты? Большинство компаний, использующих кредитные карточки, выплачивают деньги каждый календарный месяц, независимо от количества проданных вами приложений.

Каким образом будет обеспечено безопасное обслуживание кредитных карточек? Служба, которой пользуюсь я, предоставляет мне Web-страницу, на которой хранятся подробные данные о моем приложении: имя, номер версии, цена (в различных валютах) и пр. А затем я связываюсь с этой страницей из моих Web-страниц, чтобы проблемы безопасности решались службой кредитных карточек, а не мною.

Обычная почта

Некоторые покупатели приложений не станут использовать кредитные карточки, какой бы безопасной не стала Internet. Поэтому вы должны подумать о такой категории людей и позволить оплатить покупку с помощью обычной почты. Если у вашей компании есть почтовый адрес, то проблемы нет. Но если вы является разработчиком-одиночкой, позаботьтесь об использовании абонентского ящика, чтобы не делать достоянием общественности свой адрес.

Неплохо было бы включить в приложение бланк формы регистрации, заполняемый покупателем. Так вы получите всю информацию, необходимую для регистрации и генерации разблокирующих кодов. Адрес регистрации с помощью обычной почты следует поместить в один из справочных файлов, Readme-файл и на Web-узел.

Полезные советы по организации Internet-маркетинга

Ваш Web-узел должен быть информативным и максимально дружественным по отношению к пользователю. К поддержке пользователей нужно отнестись очень внимательно, и по первым же отзывам уже можно понять, на правильно ли вы пути. Ведь все мы ценим хорошее отношение, поэтому, отвечая на вопросы по электронной почте, необходимо всегда помнить, что за ними стоят конкретные люди, а не абстрактные адресаты, которые случайно загрузили ваше приложение.

Зачем платить за рекламу, если ее можно получить бесплатно? Осмотритесь вокруг, найдите напарников для взаимного рекламирования посредством баннеров, попросите друзей поместить небольшое рекламное объявление для вас и сделайте то же самое для них. Самая хорошая реклама — людская молва. Если вы позаботитесь о своих пользователях, вы скоро заметите, что желающих использовать ваш продукт гораздо больше, чем вы ожидали.

Разместите свое приложение на как можно большем количестве Web-узлов, позволяющих скачивать с них программное обеспечение. Это отнимет у вас немало времени, но окупится сполна. Всегда думайте о потребностях своих пользователей и не преувеличивайте их уровень компьютерной подготовки. Постарайтесь максимально упростить процесс регистрации своего приложения.

Не следует вступать в деловые отношения с первой попавшейся компанией, прелагающей услуги оплаты по кредитным карточкам. Сначала соберите как можно больше информации о разных компаниях (а их немало) такого рода. Всегда старайтесь найти подобную службу в своей стране. Выпуская в свет приложение, не стоит надеяться, что уже завтра вы проснетесь знаменитым, ведь на все нужно время.

Резюме

Распространение программных продуктов — довольно трудная тема. Прочитав эту главу, вы должны решить, какой путь избрать для своего приложения и как добиться, чтобы описанные здесь идеи реализовались с максимальной эффективностью.

Не пожалейте сил на проектирование хорошего Web-узла для своего приложения. Это может сэкономить много времени на поддержке, связанной с ответами на вопросы пользователей.

Разрабатывая приложение, решите, будете ли вы делать его условно-бесплатным. Об этом стоит думать наперед, если хотите “собрать хороший урожай”. Обычная почта прекрасно функционирует, и многие по-прежнему предпочитают использованию кредитной карточки в Internet традиционную отправку чеков.

Прежде чем платить деньги за рекламу, рассмотрите бесплатные методы рекламирования вашего Web-узла. Подумайте заранее о том, как будете обеспечивать зарегистрированных пользователей полноценной версией своего приложения.

Не надейтесь на быстрый успех. Должно пройти некоторое время, прежде чем о вашей программе узнают, оценят ее по достоинству и захотят купить.

Глава

29

Инсталляция и обновление программных продуктов

Саймон Ратли-Фрэйн
Дж. Алан Бруган
Йото Йотов

В главе 28 мы уже затрагивали тему распространения программных продуктов, включая способы условно-бесплатного их использования и вопросы маркетинга. В этой главе мы рассмотрим методы безотказной инсталляции и поддержки приложений, будь то условно-бесплатный вариант или коммерческий.

Мы обсудим средства инсталляции программных продуктов, а именно: использование утилиты InstallMaker (созданной компанией ClickTeam), кабинетных (`.CAB`) и информационных (`.INF`) файлов, а также утилиты InstallShield Express (ISX), которая входит в поставку C++Builder. Кроме того, мы остановимся на поддержке программных продуктов, в том числе создании и распространении новых версий программ и "заплат", а также познакомим вас с TeamSource, системой управления новыми версиями.

Установка и удаление программ

По завершении работы над приложением необходимо позаботиться о том, чтобы конечные пользователи могли без труда установить его (и все связанные с ним файлы) и чтобы после установки приложение корректно работало.

Программа инсталляции позволяет управлять составом устанавливаемых файлов и их размещением в системе, причем таким способом, что даже новичок должен без проблем установить приложение и запустить его.

Рынок предлагает множество программ, предназначенных для создания утилит установки (свободно распространяемые, условно-бесплатные и коммерческие версии). Я приведу список соответствующих Web-узлов в конце этого раздела, чтобы вы могли сами их посетить.

Генераторы программ инсталляции

Создание программы инсталляции, предназначенной для использования конечным пользователем, во многом напоминает работу в C++Builder. Вы уточняете, что именно собираетесь сделать (путем создания сценария), затем компилируете его, включаете файлы приложения, упаковываете их и создаете интерфейс пользователя. Результат этой работы представляет собой ряд файлов инсталляции, которые и подлежат распространению.

Как они работают

Некоторые программы создания утилит установки используют стиль мастера, который создает нужный сценарий; от вас требуется лишь заполнить поля соответствующими элементами, например указать включаемые файлы приложения, папку размещения приложения по умолчанию, привести информацию ознакомительного характера (Readme), а также сведения, касающиеся лицензирования продукта.

Полученная от вас информация преобразуется в сценарий, который программа-генератор использует для создания настоящих файлов инсталляции.

Файлы могут быть организованы двумя способами: в виде одного или нескольких файлов инсталляции. Результат обоих вариантов должен быть одинаков, но у каждого из них есть свои плюсы и минусы, которые обсуждаются в следующих разделах.

Однофайловый вариант инсталляции

Преимущество этого варианта состоит в том, что весь пакет инсталляции содержится в одном файле, который легко поместить на компакт-диск.

Недостаток — в невозможности распространять ваше приложение с помощью набора гибких дискет, поскольку один файл инсталляции, как правило, превышает ее объем (1,44 Мбайт).

Многофайловый вариант инсталляции

Преимущество этого типа инсталляции — в возможности распространения приложения на гибких дискетах.

Недостаток я прочувствовал лично сам. Чем больше файлов в пакете инсталляции, тем больше их может испортиться. Обиднее всего, когда портится какой-нибудь небольшой файл (размером, скажем, 2 Кбайт), а без него программа инсталляции работать уже не будет.

Программа Install Maker

Программа Install Maker от компании ClickTeam выпускается в двух версиях: коммерческой, для работы с которой нужно купить лицензию, и свободно распространяемой, которая выводит на экран для конечного пользователя сообщение о том, как программой Install Maker (от компании ClickTeam) был создан этот файл инсталляции.

Для получения дополнительной информации по лицензированию программы Install Maker обращайтесь по адресу: `http://www.clickteam.com`.

Как создать программу инсталляции с помощью Install Maker

Эту программу инсталляции можно загрузить с Web-узла компании ClickTeam или с прилагаемого к книге компакт-диска.

Install Maker прекрасно подходит для отдельных приложений, которые не требуют добавления таких более сложных средств, как записи системного реестра и возможность работы с базой данных. В таких случаях лучше использовать программу Installshield.

Install Maker создает относительно небольшую программу инсталляции с хорошим коэффициентом сжатия включаемых файлов. Мы пройдем по всем этапам этого мастер-процесса, чтобы понять, как создается программа инсталляции.

1. Открытие окна приветствия (Welcome Screen).

   После установки программы Install Maker запустите ее — к вашим услугам мастер инсталляции (Installation Wizard). На этом этапе создания программы инсталляции вы можете либо продолжить “сотрудничество” с мастером, либо использовать основной интерфейс пользователя.

   Если вы впервые работаете с этим приложением, я бы посоветовал вам пользоваться услугами мастера до тех пор, пока вы лучше не узнаете эту программу.

2. Задание исходной и приемной папок.

   Теперь вы можете выбрать папку, в которой будут храниться все ваши файлы. Имеет смысл (для данного типа инсталляции) содержать все файлы в отдельной папке. У вас также могут быть подпапки, например

   `MyApplcationFolder - MySubfolder`.

   Программа Install Maker может затем собрать все эти файлы из главной папки и подпапок (инсталлируемых с прежними именами) и установить их в желаемом каталоге на компьютере конечного пользователя. Чтобы разрешить использование подпапок, установите флажок Include Sub-Directories (Включить подпапки). Кстати, этот флажок устанавливается по умолчанию.

3. Присвоение имени программе.

   Затем укажите язык инсталлируемого приложения: English (английский) или French (французский). Разработчики программы Install Maker — французы, отсюда и такой выбор языков.

Теперь введите подходящее имя для своего приложения инсталляции. Это имя будет предъявлено конечным пользователям.

В любой момент работы с мастером вы можете щелкнуть на кнопке Preview (Предварительный просмотр), чтобы увидеть, как будет выглядеть экран, когда пользователь дойдет до этого момента инсталляции.

4. Создание оболочки.

   Найдите на этом экране основной выполняемый файл, для которого вы хотите создать пиктограмму, и присвойте ей соответствующее имя. У вас также есть возможность поместить эту пиктограмму на рабочий стол, хотя мы не советуем этого делать, поскольку слишком большое количество значков на рабочем столе только сбивает пользователя с толку.

5. Вставка дополнительной информации.

   На этом экране вы можете ввести также дополнительный текст, (например, содержимое файла `Readme` или информацию о лицензировании). В данном разделе не предусмотрена кнопка открытия файла, но вам предлагается ввести текст в подготовленное для этого поле методом копирования и вставки. После ввода дополнительной информации желательно просмотреть весь текст, чтобы убедиться в его корректном форматировании (для этого щелкните на кнопке Preview).

6. Настройка окна инсталляции.

   На этом этапе можно выбрать желаемый размер экрана инсталляции и имя, которое будет видеть конечный пользователь при работе с программой инсталляции.

7. Задание растровых изображений для инсталляции.

   В этой части мастер позволяет разместить в программе инсталляции два растровых изображения. Небольшое предупреждение: любые дополнительные файлы увеличивают общий размер программы инсталляции. Безусловно, это не проблема, если вы собираетесь распространять свое приложение на компакт-дисках, но если вы планируете использовать для этого Internet или гибкий диск (диски), конечный файл следует сделать как можно меньше (чем он меньше, тем привлекательнее для потенциальных покупателей).

   Первое изображение, которое вы можете выбрать, должно иметь определенные размеры (128 пикселей в ширину и 280 пикселей в высоту) и использовать не более 256 цветов. Это изображение будет показано на главном вводном экране и располагаться слева от текста. Если вы плохо себе представляете эту картину, выберите любое изображение и щелкните на кнопке Preview.

   Второе изображение будет использовано для фонового покрытия главного экрана и не должно содержать более 256 цветов. Чтобы посмотреть, как это будет выглядеть, щелкните на кнопке Preview.

8. Задание каталога инсталляции по умолчанию.

   Следующий экран позволяет ввести имя каталога по умолчанию, в который ваше приложение и связанные с ним файлы будут инсталлированы на компьютере конечного пользователя. Вот пример.

   ```
   "C:\Program Files\MyCompany\MyApplcation"
   ```

   Это не что иное, как забота о начинающем компьютерном пользователе, которому для успешной установки приложения достаточно лишь щелкать на кнопке Next.

Путь, предлагаемый по умолчанию, может включать любое действительное устройство и папку; но вам вряд ли стоит вносить в него изменения.

Этот мастер также предоставляет возможность считывания каталога инсталляции из `*.ini`-файла или записи системного реестра.

9. Установка заключительных параметров (End Page Options).

В этом разделе мастера можно указать файл Readme или лицензионное соглашение, которое конечный пользователь сможет прочитать по завершении работы программы инсталляции.

Можно также сделать так, что по завершении программы инсталляции будет запускаться файл приложения.

10. Создание программы удаления приложения (Uninstall Program).

Следующий экран предлагает принять решение только по одной опции — созданию программы удаления приложения. Этот флажок следует устанавливать практически всегда (он как раз и установлен по умолчанию), если не существует каких-либо исключительных обстоятельств для принятия обратного решения, поскольку конечный пользователь должен всегда иметь возможность при необходимости удалить приложение, причем самым простым образом.

Итак, если эта опция выбрана, программа инсталляции создаст небольшой `.exe`-файл, который пользователь сам сможет запустить для удаления приложения. Это можно сделать с помощью ярлыка, автоматически создаваемого во время инсталляции, или посредством опции Установка и удаление программ системной панели управления.

11. Завершение работы мастера.

Мы подходим к последнему экрану мастера, который позволяет сделать окончательный выбор — создавать программу инсталляции или нет. Если вы уверены в корректной установке всех параметров, смело щелкните на кнопке Finish (Готово), запустив тем самым механизм построения программы инсталляции. Если же вы не вполне уверены в корректной установке каких-либо параметров, просто щелкните на кнопке Back (Назад), чтобы еще раз перепроверить интересующие вас параметры.

12. Построение программы инсталляции.

Убедившись в вашем решении приступить к построению программы инсталляции, мастер предложит указать путь и имя для создаваемого `.exe`-файла. Поступайте по своему усмотрению, но желательно не помещать его в одну папку с основными файлами вашего приложения; в противном случае этот файл будет добавлен в программу инсталляции, если вы решить переделать ее.

Когда этот файл будет создан, желательно сохранить файлы своего проекта в безопасном месте на случай возможных будущих обновлений и исправлений.

Весьма желательно протестировать новоиспеченную программу инсталляции. Для этого лучше всего вызвать Проводник Windows и запустить этот файл, что и должен будет сделать конечный пользователь. После внесения всех необходимых изменений и доработок, т.е. когда вы будете полностью удовлетворены работой программы инсталляции, следует обязательно запустить ее на другом компьютере, чтобы убедиться в корректности установки приложения и работы всех его функций.

Не стоит расстраиваться, если не все получилось с первого раза; просто вернитесь назад и внесите изменения в файлы проекта, а затем снова постройте программу инсталляции и снова ее протестируйте. Но сначала не забудьте удалить текущую версию приложения.

13. Создание дополнительных пиктограмм и дополнительных файлов.

    По окончании сотрудничества с мастером вы можете создать дополнительную пиктограмму для своего приложения в главном интерфейсе пользователя. Кроме того, возможно, вы захотите иметь отдельную пиктограмму для справочного файла или файла `Readme.txt`, если таковые вами предусмотрены.

    Для этого выберите вкладку Files, а затем — файл, для которого собираетесь создать пиктограмму. В правой части экрана есть два поля Folder (Папка) и Name (Имя). Укажите папку, в которой желаете установить пиктограмму, и присвойте ей подходящее имя.

На заметку Чтобы установить новую пиктограмму в одну папку с пиктограммами, сгенерированными автоматически, используйте в качестве главного `.exe`-файла то же имя папки.

Удаление приложения

Программа инсталляции должна предусматривать возможность для конечного пользователя удалить приложение. В большинство генераторов программ инсталляции это средство уже встроено и, подобно Install Maker, устанавливается автоматически, поэтому вам не нужно даже беспокоиться о включении или добавлении этой функции.

К сожалению, пользователи, установившие ваше приложение, захотят удалить его из своей системы. Поэтому вы должны быть уверены в том, что они смогут легко от него избавиться и по возможности полностью.

Общеизвестно, что многие приложения, которые вы устанавливаете в своей системе, не могут полностью выполнить "самоликвидацию", поэтому возникает необходимость в программах Windows Cleaner. Но вы как разработчик просто обязаны убедиться, что после удаления приложений на жестких дисках пользователей не останется "забытых" файлов или записи в системном реестре.

Казалось бы, для этого достаточно создать список всех файлов, необходимых для успешной работы приложения на другом компьютере, а затем убедиться в том, что все они удалены.

Чтобы реализовать такую проверку, запустите свою программу инсталляции, а по ее завершении выполните процедуру удаления, чтобы узнать, остались ли неудаленные файлы. Присутствие "остаточных" файлов возможно в том случае, если пользователь добавил в каталог инсталляции свои файлы (или они были созданы во время работы с вашим приложением) либо у вас не была автоматически установлена опция удаления приложения.

Очень трудно (практически невозможно) добиться "чистоты" удаления путем автоматического создания средства удаления генератором программы инсталляции. Дело в том, что это средство удалит лишь файлы и записи в системном реестре, установленные при запуске программы инсталляции. Это средство не в состоянии удалить ничего, что было создано во время работы вашего приложения.

Такой же результат может быть получен и при работе с другими приложениями, которые пытались самоликвидироваться. В процессе удаления приложения программа удаления может сообщить о том, что папка *xx* не может быть удалена. Причиной генерации такого сообщения может оказаться присутствие в этой папке "чужих" файлов, которых не было при первоначальной установке приложения и которые были созданы во время его работы.

Иногда среди оставшихся файлов могут оказаться такие, которые и должны остаться на компьютере конечного пользователя. Например, ваше приложение может использовать некоторые DLL-файлы, которые необходимы системе Windows для продолжения работы. В такой

ситуации пользователь увидит сообщение о том, что не удален системный файл, и ему будет предложено подтвердить намерение его удалить.

Ниже перечислены адреса Internet, имеющие отношение к другим средствам создания программ инсталляции.

Setup Factory: `http://www.indigorose.com/`
Inno Setup: `http://www.jordanr.dhs.org/`
Soysal Setup: `http://www.soysal.com/ssetup`
Wise Setup: `http://www.wisesolutions.com/imakfeat.htm`

CAB- и INF-файлы

Кабинетные и информационные файлы представляют собой эффективный способ распространения и программных продуктов и Internet-приложений, позволяющий сократить время загрузки. Их бесшовная интеграция с Web-броузером Microsoft Internet Explorer позволяет автоматически выполнить инсталляцию файлов и обеспечивает управление версиями. В следующих разделах вы узнаете, как создать собственные CAB- и INF-файлы.

Немного о CAB-файлах

Когда речь идет о хранении или передаче файлов, никому не хочется иметь дело с большим объемом данных. Решение этой проблемы — в так называемом сжатии файлов. Программы сжатия файлов основаны на том, что в последовательностях нулей и единиц существуют часто повторяющиеся фрагменты. Этим фрагментам присваиваются специальные коды, в результате чего большие файлы преобразуются в файлы меньшего размера.

Компания Microsoft при сжатии файлов использует формат CAB (сокращение от *Cabinet*), который опирается на Lempel-Ziv-сжатие. Как Microsoft Windows, так и Microsoft Office распространяются с использованием пакетов CAB-инсталляции, как показано на рис. 29.1. Закономерный вопрос: почему нам (разработчикам) желательно использовать этот формат?

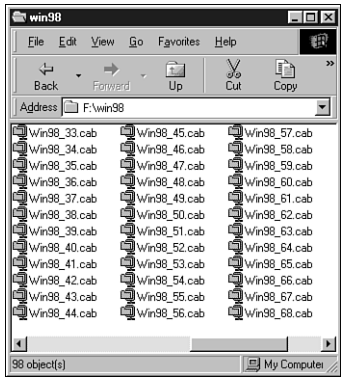


Рис. 29.1. Инсталляция Microsoft Windows с использованием CAB-пакетов

Помимо уменьшения времени загрузки, что является безусловной целью всех программ сжатия, CAB-файлы в настоящее время представляют собой самый простой способ сжатия и распространения элементов управления ActiveX и классов Java через Internet. Кроме того, компания Microsoft в своем наборе инструментальных средств разработки программного обеспечения, именуемом Cabinet Software Development Kit (SDK), обеспечивает разработчиков необходимыми компонентами и инструментарием для применения этой технологии.

В папке `BIN` C++Builder вы найдете утилиту CABARC, которая является частью Cabinet SDK. К сожалению, другие полезные программы, о которых пойдет речь ниже в этой главе, не поставляются в комплекте с C++Builder. Поэтому, прежде чем продолжить, вам следует загрузить последнюю версию Cabinet SDK, воспользовавшись адресом: `http://msdn.microsoft.com/workshop/management/cab/cabdl.asp`.

Создание и выделение CAB-файлов

Набор Cabinet SDK поставляется с тремя следующими утилитами, которые хранятся в каталоге CABINET SDK\BIN .

- EXTRACT.EXE. Простая утилита выделения.
- CABARC.EXE. Базовая утилита создания/выделения/просмотра.
- MAKECAB.EXE. Мощная утилита установки/управления/создания/выделения/просмотра. Использует управляющие файлы.

Как видите, вам предлагается многофункциональные средства. В целях демонстрации я буду использовать утилиту CABARC, поскольку с ее помощью можно выполнить все необходимые задачи (она используется средой (IDE) C++Builder).

Чтобы узнать диапазон возможностей этой утилиты, введите в командной строке имя CABARC. Результат выполнения этой команды приведен в листинге 29.1.

Листинг 29.1. Опции команды CABARC

```
Microsoft (R) Cabinet Tool - Version 1.00.0601 (03/18/97)
Copyright (c) Microsoft Corp 1996-1997. All rights reserved.

Usage: CABARC [опции] команда cab-файл [@list] [files] [dest_dir]

Команды:
   L   Перечислить содержимое cab-файла (например, cabarc l test.cab)
   N   Создать новый cab-файл (например, cabarc n test.cab *.c app.mak *.h)
   X   Выделить файл (файлы) из cab-файла (например, cabarc x test.cab foo*.c)

Опции:
  -c   Подтвердить обработку файлов.
  -o   При выделении перезаписывать без запроса на подтверждение.
  -m   Установить тип сжатия [LZX:<15..21>|MSZIP|NONE], (по умолчанию MSZIP).
  -p   Сохранить имена путей (абсолютные имена не разрешены).
  -P   Удалять заданный префикс файлов при добавлении.
  -r   Заходить в папки при добавлении файлов (см. также -p).
  -s   Зарезервировать место в cab-файле для подписи (например,
                                                    -s 6144 reserves 6K byte)
  -i   Установить ID набора cab-файлов при их создании (по умолчанию 0)

Примечания
----------
При создании cab-файла знак "плюс" (+) можно использовать
в качестве имени файла для указания границы папки; например,
                            cabarc n test.cab *.c test.h + *.bmp

При выделении файлов на диске элемент <dest_dir>, если он указан,
должен оканчиваться обратной косой чертой; например,
                        cabarc x test.cab bar*.cpp *.h d:\ test\

Опцию -P (удалить префикс) можно использовать для удаления
информации о пути, например
```

```
        cabarc -r -p -P myproj\  a test.cab myproj\ balloon\ *.*
Опцию -P для удаления нескольких путей можно использовать несколько раз.
```

Для более эффективного освоения утилит Cabinet SDK советуем прочитать соответствующую документацию (ее можно найти в папке \DOCS), где приведено множество примеров с подробными разъяснениями.

По сути, сжатие и распаковка выполняются путем задания соответствующих опций. Например, чтобы создать CAB-файл, содержащий все DLL-библиотеки в текущем каталоге, введите следующую команду.

```
cabarc n newcab.cab *.dll
```

Для выделения этих DLL в другую папку введите следующую команду.

```
cabarc x newcab.cab *.dll c:\windows\desktop\
```

При распаковке CAB-файлов не забудьте поместить обратную черту в конец папки приемника. В предыдущем примере команда `cabarc x newcab.cab *.dll c:\windows\desktop` (т.е. без обратной черты в конце папки приемника) поместила бы все выделенные DLL-библиотеки в текущий каталог.

Создание самораспаковывающихся CAB-файлов

Как видите, при наличии соответствующего инструментария создание и выделение CAB-файлов — вполне простая задача. Однако попробуйте представить следующий сценарий: у вас есть небольшой CAB-файл, который вы собираетесь распространять через Internet. Но вряд ли при этом вы захотите утомлять своих пользователей загрузкой и изучением программ сторонних производителей. Не лучше ли создать `.exe`-файл, который сам распакует свое содержимое в текущую папку?

Оказывается, это чрезвычайно просто сделать, выполнив следующее.

```
copy /b extract.exe+newcab.cab newcab.exe
```

Если теперь запустить файл `newcab.exe`, он сам распакуется без каких-либо усилий с вашей стороны.

Немного об INF-файлах

Информационные (INF-) файлы — это просто текстовые файлы, содержащие инструкции по установке драйверов устройств, программных приложений или компонентов Internet. Конечно, не все INF-файлы содержат такие сложные инструкции по инсталляции — некоторые из них просто модифицируют системный реестр.

INF-файлы организованы по разделам. Каждый раздел содержит специальные записи и директивы, используемые для копирования файлов, установки драйверов или выполнения любых других задач. Ниже мы остановимся на архитектуре INF-файлов и наиболее часто используемых директивах.

Не путайте INF-файлы с INI-файлами. Они имеют похожий синтаксис, но файлы инициализации (INI) используются только для хранения данных конфигурации.

Разделы и директивы INF-файлов

Программа установки Windows (Setup) находит разделы по их именам. Следовательно, порядок разделов значения не имеет. Имя раздела должно быть заключено в квадратные скобки ([]) и не превышать 28 символов, как показано в этом примере.

```
[Strings]
Msg="Welcome"
```

На заметку В именах разделов, записях и директивах не различаются прописные и строчные буквы. Если информационный файл содержит два раздела с одинаковыми именами, программа Setup объединит их содержимое в один раздел.

В табл. 29.1 и 29.2 описаны наиболее популярные разделы и директивы INF-файла.

Таблица 29.1. INF-разделы

Раздел	Описание
`ClassInstall32`	Инициализирует и инсталлирует классы установки устройств
`ControlFlags`	Определяет, отображается ли данная модель/устройство в окне мастера Add New Hardware Wizard (Добавить новое оборудование)
`DestinationDirs`	Определяет папку приемника при операциях копирования, удаления или переименования
`InterfaceInstall32`	Инициализирует и устанавливает классы интерфейсов с устройствами
`Manufacturer`	Идентифицирует изготовителя устройств
`SourceDisksFiles`	Задает местоположение файлов, подлежащих инсталляции
`SourceDisksNames`	Задает дистрибутивные диски для инсталляции
`Strings`	Определяет все лексемы `strkey`, используемые в INF-файле

Таблица 29.2. INF-директивы

Директива	Описание
`AddInterface`	Инсталлирует поддержку для интерфейсов устройств
`AddReg`	Добавляет/модифицирует подключи и записи системного реестра
`AddService`	Инсталлирует службы драйвера устройства
`BitReg`	Устанавливает биты в записях `REG_BINARY` системного реестра
`CopyFiles`	Копирует файлы с дистрибутивных дисков в папку приемника
`DelFiles`	Удаляет файлы из папки приемника
`DelReg`	Удаляет подключи и записи системного реестра
`DelService`	Удаляет существующие службы
`Ini2Reg`	Перемещает строки из существующих INI-файлов в системный реестр
`LogConfig`	Информирует о наборе ресурсов конфигурации оборудования
`RenFiles`	Переименовывает файлы в папке приемника
`UpdateIniFields`	Информирует о разделах, в которых модифицированы существующие INI-файлы
`UpdateInis`	Читает/записывает строки в существующие INI-файлы

В этих двух таблицах приведены лишь общие сведения о разделах и директивах INF-файлов. Более подробное описание можно найти в комплекте документации Windows по разработке драйверов (Windows Driver Development Kit — DDK), к которому можно получить доступ по адресу: `http://www.microsoft.com/ddk/`.

INF-файлы позволяют включать комментарии. Символы, следующие за точкой с запятой (`;`) в той же строке, рассматриваются как комментарии (если они не заключены в кавычки).

Создание INF-файлов

Теперь мы готовы создать INF-файл. Файл, который мы будем создавать (`Sample.inf`), находится в папке `INF Files` прилагаемого к книге компакт-диска.

Как упоминалось выше, информационные сценарии представляют собой простые текстовые файлы, поэтому для их создания можно использовать любой текстовый процессор — лично я предпочитаю C++Builder. Из меню C++Builder выберите команду File⇨New и щелкните на объекте Text. Сохраните новый файл под именем `Sample.inf`.

В вашем сценарии инсталляции должно быть предусмотрено выполнение следующих задач.

- Копирование всех программных файлов (в данном случае `PROGRAM.EXE` и `PROGRAM.TXT`) в папку `Windows`.
- Модификация реестра, чтобы файл `PROGRAM.EXE` автоматически запускался при загрузке Windows.

Как вы, вероятно, уже догадались, на первом этапе для передачи файлов в папку приемника используется директива `CopyFiles`. Ее синтаксис имеет следующий вид.

```
Copyfiles=@filename | file-list-section[, file-list-section]...
```

Чтобы скопировать один файл, укажите его имя, предварив его символом `@`. Чтобы скопировать список файлов, создайте новый раздел и задайте имя раздела, как в этом примере.

```
CopyFiles=Sample.Copy

[Sample.Copy]
Program.exe
Program.txt
```

К сожалению, здесь отсутствует важная информация — папка-источник и папка-приемник. Из табл. 29.1 следует, что эти данные содержатся в разделах `DestinationDirs`, `SourceDisksNames` и `SourceDisksFiles`.

```
[DestinationDirs]
Sample.Copy=10

[SourceDisksNames]
1="Drive D:"

[SourceDisksFiles]
Program.exe=1
Program. txt=1
```

На заметку

При задании папки-приемника можно использовать такие встроенные значения, как 24 (папка Applications), 11 (папка System), 30 (папка Root загрузочного диска) и пр. В предыдущем примере число 10 означает использование папки Windows. (Как упоминалось выше, описание каждого значения можно найти в электронной документации DDK.)

Если нужно создать новую папку-приемник, объедините эти установленные значения с вашим строковым значением. Например, для создания папки MyProgram на загрузочном дисковом устройстве пользователя введите следующее.

```
Sample.Copy=30,"MyProgram"
```

Теперь можно перейти к следующему этапу — автозапуску программы Program.exe при загрузке Windows. Это делается путем добавления пути к этой программе в ключевую запись HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Run системного реестра. Для выполнения этой задачи служит директива AddReg.

```
AddReg=add-registry-section[, add-registry-section] ...

[add-registry-section]
reg-root, [subkey], [value-entry-name], [flags], [value]
```

Возможные значения элемента reg-root приведены в табл. 29.3.

Таблица 29.3. Значения корневого ключа реестра

Значение	Описание
HKCR	Аббревиатура от HKEY_CLASSES_ROOT
HKCU	Аббревиатура от HKEY_CURRENT_USER
HKLM	Аббревиатура от HKEY_LOCAL_MACHINE
HKU	Аббревиатура от HKEY_USERS

Параметры subkey и value определяют, какие ключи или значения системного реестра должны быть созданы. Рассмотрим пример.

```
AddReg=Sample.Reg

[Sample.Reg]
HKLM,"Software\Microsoft\Windows\CurrentVersion\Run",
↳"SampleProgram",,"%10%\Program.exe "
```

Обратите внимание, как встроенное значение, заключенное в символы процентов (%), дополняется строковым значением (в данном случае подразумевается использование папки Windows).

У нас пока отсутствуют два раздела. Первый — Version — задает подпись. По умолчанию в качестве подписи используется значение $CHICAGO$ (Chicago — это название бета-версии Windows 95, которое вовсе не означает, что ваши сценарии не будут совместимы с Windows NT или с любой 32-разрядной версией Windows). Второй и самый важный раздел — DefaultInstall — включает код, который будет выполняться при инсталляции INF-файла. Окончательная версия файла Sample.inf содержится в листинге 29.2.

Листинг 29.2. Файл Sample.inf

```
[Version]
Signature="$CHICAGO$"
```

```
[DefaultInstall]
AddReg=Sample.Reg
CopyFiles=Sample.Copy

[Sample.Copy]
Program.exe
Program.txt

[DestinationDirs]
Sample.Copy=10

[SourceDisksNames]
1="Drive D:",,,

[SourceDisksFiles]
Program.exe=1
Program.txt=1

[Sample.Reg]
HKLM,"Software\Microsoft\Windows\CurrentVersion\Run",
↳"SampleProgram",,"%10%\ Program.exe"
```

Чтобы протестировать сценарий, скопируйте файлы `Sample.inf`, `PROGRAM.EXE` и `PROGRAM.TXT` на дискету или в корневую папку жесткого диска (я использую диск `D:`), щелкните правой кнопкой мыши на пиктограмме `Sample.inf` и выберите из контекстного меню команду Install. На рис. 29.2 показано, что в системный реестр добавлено новое значение.

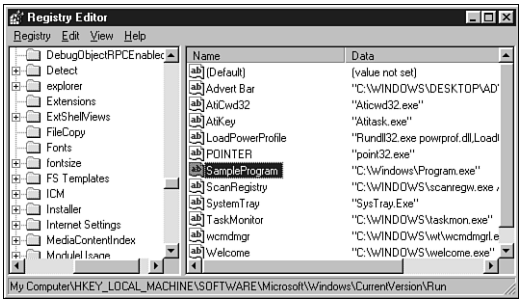


Рис. 29.2. Сценарием инсталляции в системный реестр добавлено новое значение

Немного о пакетах Internet

Предположим, вы создали элемент управления ActiveX или выполняемый файл ActiveX и теперь хотели бы сделать его доступным для загрузки. Как обеспечить пользователям возможность инсталлировать свое Internet-приложение прямо из Web-узла?

Разработчики Microsoft предусмотрели простое решение — сочетание мощи INF- и CAB-файлов, именуемое Internet-пакетами. Элемент управления OCX и все необходимые файлы сжимаются в один или несколько CAB-пакетов. Затем INF-файл направляет ваш броузер туда, где находятся эти пакеты, и инструктирует о способе их инсталляции.

На момент написания этой книги Microsoft Internet Explorer (версия 3.x или более поздние) был единственным Web-броузером, который поддерживал CAB-ориентированные Internet-пакеты.

Суть Internet-пакета

В главе 22 вы узнали о том, как разворачивать элементы управления и формы ActiveX с помощью Web-параметров развертывания C++Builder. Теперь вы познакомились с INF- и CAB-файлами, и краткий обзор этих средств развертывания поможет вам понять, как работают Internet-пакеты.

В целях демонстрации я создал и развернул небольшой проект ActiveForm (рис. 29.3). Прежде всего рассмотрим, как пакетированные элементы управления ActiveX встраиваются в Web-страницу. Листинг 29.3 содержит исходный HTML-код Web-страницы, сгенерированный средствами C++Builder.

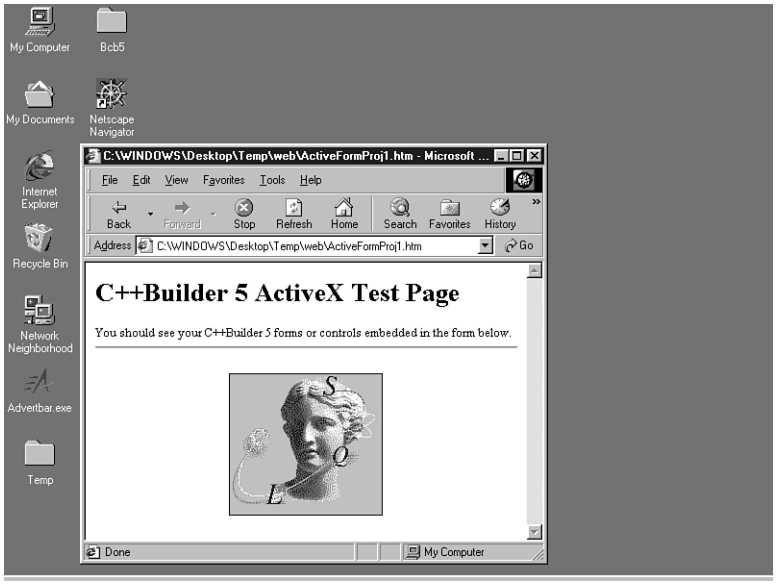


Рис. 29.3. Проект `ActiveFormProject1`*, развернутый в Internet Explorer*

Листинг 29.3. `ACTIVEFORMPROJ1.HTM`

```
<HTML>
<H1> C++Builder 5 ActiveX Test Page </H1><p>
Вы должны увидеть свои формы или элементы управления C++Builder
↳5, встроенные в эту форму.
<HR><center><P>
<OBJECT
      classid="clsid:FE0A6145-3478-11D4-B120-0080C8DF95D0"
      codebase="C:\WINDOWS\Desktop\Temp\web\ActiveFormProj1.inf"
      width=160
      height=148
```

```
      align=center
      hspace=0
      vspace=0
>
</OBJECT>
</HTML>
```

Очевидно, формы ActiveX встраиваются в Web-страницы с использованием тега `OBJECT`. Помимо описания размеров и способа выравнивания элемента управления, этот тег содержит два важных атрибута — `CLASSID` и `CODEBASE`. Атрибут `CLASSID` задает уникальный идентификатор класса, под которым ваш объект был зарегистрирован в реестре Windows. Атрибут `CODEBASE` означает местоположение и (необязательно) версию файла, содержащего объект.

Как показано в листинге 29.3, атрибут `CODEBASE` указывает на INF-файл. Однако предусмотрена поддержка и других типов файлов. Например, если у вас есть один элемент управления ActiveX, который не требует никаких поддерживающих библиотек DLL, вы можете направить атрибут `CODEBASE` прямо на OCX-файл, как показано ниже.

```
codebase=
↳"C:\WINDOWS\Desktop\Temp\web\ActiveFormProj1.ocx#version=
↳1,0,0,0"
```

Другое, даже более удачное решение — указать местоположение CAB-файла. Этот CAB-пакет должен содержать ваши INF-, OCX- и при необходимости DLL-файлы.

```
codebase=
↳"C:\WINDOWS\Desktop\Temp\web\ActiveFormProj1.cab#version=
↳1,0,0,0"
```

Теперь заглянем в самую суть вашего Internet-пакета — информационный файл (листинг 29.4).

Листинг 29.4. `ACTIVEFORMPROJ1.INF`

```
;C++Builder-generated INF file for ActiveFormProj1.ocx
[Add.Code]
ActiveFormProj1.ocx=ActiveFormProj1.ocx
VCL50.bpl=VCL50.bpl

[ActiveFormProj1.ocx]
file=C:\ WINDOWS\ Desktop\ Temp\ web\ ActiveFormProj1.cab
clsid={ FE0A6145-3478-11D4-B120-0080C8DF95D0}
RegisterServer=yes
FileVersion=1,0,0,0

[VCL50.bpl]
file=
FileVersion=5,0,6, 18
DestDir=11
```

Оказалось проще, чем вы думали, не так ли? INF-файл содержит список файлов, подлежащих инсталляции. Каждый файл описан в своем разделе. В каждом разделе вы найдете местоположение CAB-пакета, версию файла и папку приемника. Например, папка приемника для файла `VCL50.BPL` задана значением 11. Вспомните встроенные значения, о которых мы упоминали выше. Так вот, число 11 означает папку Windows `System`.

Версии, обновления и заплаты

В этом разделе мы поговорим об управлении версиями и о реализации заплат и обновлений. Заплаты и обновления — это логическое развитие вашего приложения; благодаря им конечные пользователи могут поддерживать самые последние версии вашего приложения.

Версии

Понятие версий программных продуктов имеет большое значение; они позволяют держать конечных пользователей в курсе на предмет "новизны" используемого ими продукта и интересоваться существованием и возможностью использования более новых его версий. Версия также позволяет вам оказывать более точную техническую поддержку. Например, некоторая функция в версии 1.1 может не быть доступной в версии 1.0, или ошибка, присутствующая в версии 1.0, может быть исправлена в версии 1.1.

Поддержка версий

Вы можете реализовать собственную форму управления версиями путем добавления номера версии в поле AboutBox вашего приложения или позволить C++Builder сделать это за вас — для этого выберите команду Project⇨Options (Проект⇨Параметры), а затем активизируйте вкладку Version Info (Информация о версиях). Здесь можно установить опцию, которая позволит C++Builder добавлять в создаваемое приложение информацию о его версии. Эта информация включает такие понятия, как Major Number (основной номер версии), Minor Number (дополнительный номер версии), Release Number (номер выпуска) и Build Number (номер создаваемого проекта).

Можно настроить C++Builder на работу в автоинкрементном режиме, чтобы номер создаваемого приложения автоматически увеличивался на единицу при каждом создании приложения.

Вкладка Version Info также позволяет вводить информацию о создаваемом вами файле (например, информацию о вашей компании, описание файла и уведомление об авторском праве).

Вся эта информация будет показана конечному пользователю, если он щелкнет правой кнопкой мыши на имени файла.

В качестве альтернативного варианта можно использовать какой-нибудь компонент стороннего производителя, чтобы сначала сохранить информацию о версии, а затем дать возможность пользователям просматривать эту информацию в приложении.

Конечно, вы вряд ли захотите ставить пользователя перед необходимостью открывать Windows Explorer, а затем щелкать правой кнопкой мыши на имени файла (при условии, если таковой будет найден). Поэтому самый простой способ обеспечить пользователя информацией о версии приложения — поместить строку в поле AboutBox со сведениями о версии, с которой он работает. Предлагаем один из возможных вариантов такой информации.

```
MyApplication
Version 1.1
```

Затем, когда вы в очередной раз обновите файл, можно увеличить номер версии до значения 1.2 и т.д. У вас есть также возможность использовать более длинный номер версии, чтобы слишком частые обновления не вынуждали вас изменять основной номер версии приложения.

```
MyApplcation
Version 1.1.0.0
```

В этом случае при рядовом обновлении (например, исправлении ошибки) мы просто инкрементируем последнюю цифру.

```
MyApplication
Version 1.1.0.1
```

При таком подходе вся основная документация и имена программ останутся прежними, что сэкономит уйму времени при создании Web-страниц и пр., зато конечные пользователи будут твердо знать, что обладают самой последней версией.

Еще один важный элемент управления версиями — регистрация обновления. Для этого вполне подойдет текстовый файл, размещенный в одной папке с остальными файлами проекта.

Информация, содержащаяся в этом документе, должна включать номер версии и изменения, внесенные в файл проекта. Вот пример.

```
1.1.0.0 Первая опубликованная версия.
------
1.1.0.1 Исправлена ошибка форматирования шрифтов.
--------
Изменено растровое изображение основного экрана.
```

Таким образом можно отслеживать все изменения, вносимые в приложение, и, получив вопрос по части технической поддержки приложения, вы сразу поймете, была уже замечена и исправлена в версии пользователя ошибка, или здесь что-то иное.

Обновления для усовершенствования приложений

Под обновлениями понимают неизбежные усовершенствования приложений. Как лучше всего реализовать обновления — решать вам, а наша задача — предложить на выбор несколько методов.

Усовершенствование приложений

Любое изменение в приложении квалифицируется как обновление приложения. Но вы должны решить, стоит ли делать обновление доступным для конечных пользователей или подождать других усовершенствований, прежде чем выпускать следующую версию. Мы не можем сказать, когда следует выпускать обновления, поскольку это определяется многими факторами. Серьезные ошибки должны быть исправлены немедленно, чтобы другие пользователи не сталкивались с теми же проблемами. Следующее, над чем стоит подумать, — как оповестить пользователя, что вы выпустили обновленную версию.

Методы уведомления об обновлениях

Если вы относите свое приложение к разряду условно-бесплатных, то уведомление ваших клиентов о его обновлении не составляет проблемы, поскольку в вашем компьютере должны храниться данные обо всех покупателях вашего продукта. Вы могли бы создать приложение со всеми их адресами электронной почты и по мере необходимости посылать им уведомления о возможности получить обновленный вариант приложения.

Первый метод — самый простой. Несложная база данных с компонентом электронной почты способна кардинально упростить отправку целой пачки почтовых сообщений.

Для своих условно-бесплатных приложений вы можете построить базу данных и на одной вкладке поместить записи о покупателях, а на другой — указать приложения и информацию, связанную с покупкой продукта (например, дату покупки и пр.). Затем можно использовать компонент `TMemo` для записи адресов электронной почты каждого покупателя и компонент

TEmail для передачи почты из приложения регистрации в стандартную программу работы с электронной почтой, готовую к отправке.

Я использую компонент электронной почты от фирмы AHM Triton Tools (www.tritontools.com) для отправки сообщений из моего приложения регистрации, но существует множество свободно распространяемых компонентов для отправки почтовых сообщений посредством программ MAPI или SMTP. Следует отметить, что компонент TEmail (http://www.econos.de/) на момент написания еще не был доступен для работы в среде C++Builder 5.

В приложении регистрации предусмотрите обход записей всех пользователей, возьмите адрес электронной почты из базы данных записей и текст сообщения из компонента TMemo, а затем разошлите по очереди сообщения всем адресатам. Такой подход избавит вас от необходимости готовить сотни отдельных посланий или вносить адрес каждого покупателя в соответствующее поле универсального сообщения для последующей отправки по электронной почте.

Другой способ информирования покупателей вашего продукта о наличии его обновленной версии — поместить информацию об обновлении на "видное место" вашей Web-страницы, если таковая существует. Единственный недостаток такого решения в том, что вы полагаетесь на желание (или возможность) пользователей посещать ваш Web-узел вообще и на предмет получения информации об обновлениях в частности.

Еще один метод называется Listbot (см. http://www.listbot.com). Он предусматривает использование бесплатной службы электронной почты, к которой могут подключиться покупатели ваших продуктов. Это позволит посылать сообщения каждому клиенту, подключившемуся к этой службе. Идея сама по себе неплоха, но снова приходится рассчитывать на то, что покупатели ваших продуктов ее поддержат. В противном случае информация об обновлениях пройдет мимо них.

Единственный надежный способ информирования покупателей о наличии обновленных версий — послать им персональное сообщение по электронной почте.

Реализация обновлений

После того как вы подготовите обновленную версию приложения к отправке пользователям (или сделаете ее пригодной для загрузки), вам следует решить, каким образом сделать это обновление доступным для них. Можно создать так называемую *заплату* или "подать" целое приложение в виде новой программы инсталляции.

Когда стоит прибегать к новой загрузке

Иногда конечному пользователю лучше предоставить новый файл инсталляции — это позволит быть уверенным, что программа инсталляции перезапишет или обновит текущие файлы.

Конечный пользователь, который уже использовал вашу программу инсталляции для первоначальной установки приложения, не будет беспокоиться о новых программах.

Один недостаток большинства разработчиков состоит в том, что они забывают, что рядовой пользователь не так хорошо знает компьютер, как они. Разработчики "по инерции" распространяют свои знания на всех остальных. Поэтому, завершая работу над приложением, вам не помешает мысленно сделать несколько шагов назад и взглянуть на свое приложение и программу инсталляции глазами новичка.

Заплаты

Заплаты — еще один способ обновления программ. Они позволяют сохранить исходный файл приложения, а затем отредактировать его программным путем, получив тем самым обновленную версию приложения.

Чтобы создать заплату, вы должны найти способ тщательно просмотреть исходный файл приложения, а затем отыскать различия в ANSI-коде между этим и новым файлами. Затем заплата может создать сценарий редактирования для преобразования исходного файла в обновленную версию.

Для выполнения этой задачи есть прекрасная программа Patch Maker от фирмы Click Team. Программа Patch Maker распространяется в виде коммерческой и бесплатной версий, и вы можете загрузить ее с Web-узла (`http://www.clickteam.com`) или с компакт-диска, прилагаемого к книге.

Создание заплаты

Если вы решили передать пользователям программную заплату, то должны ее создать. Свободно распространяемая версия программы Patch Maker от фирмы Click Team прекрасно справляется с этой работой. (Patch Maker рассматривается в следующем разделе).

Заплаты "осматривают" текущую версию файла приложения, а затем, если он корректен, обновляет его путем редактирования файла до получения обновленного варианта.

При этом важно отслеживать версии приложения, поскольку заплата может "залатать" только определенную версию. Рассмотрим пример.

Первая версия	1.0
Версия с исправленной ошибкой	1.1
Усовершенствованная версия	1.2

Если вы создадите заплату для перехода от версии 1.1 к версии 1.2, то ее нельзя применять к версии 1.0, чтобы получить версию 1.2. Если вы все же попытаетесь это сделать, программа заплаты возвратит признак ошибки.

Программа Patch Maker

Если вы хотите успешно использовать программу Patch Maker, то после ее установки вам придется выполнить определенную подготовительную работу.

Прежде всего вам понадобится папка, в которую следует поместить только исходный файл приложения. Затем создайте еще одну папку с единственным файлом — новым файлом приложения. Дело в том, что Patch Maker запросит имена этих папок во время настройки мастера заплат.

Несмотря на то что Patch Maker может создавать заплату для нескольких файлов сразу, мы остановимся на заплате для одного файла, поскольку чаще всего одно обновление связывается с одним файлом. Однако ничто не помешает вам, поэкспериментировав, создать заплату для нескольких файлов.

После создания обновленного файла запустите программу Patch Maker. Тем самым вы активизируете мастера заплат (Patch Wizard), который без промедления готов приступить к созданию заплаты для вашего приложения.

1. Первый экран мастера заплат (Welcome Screen).

   На этом экране у вас есть возможность продолжить работу с мастером или использовать главный интерфейс пользователя для создания программы инсталляции.

   Если вы впервые используете это приложение, я бы посоветовал вам работать с мастером до тех пор, пока не узнаете эту программу как свои пять пальцев.

2. Задание папок источника и приемника.

   На этом экране вам предлагается ввести имена двух папок:

- папки со старой версией файла, т.е. файла, подлежащего обновлению;
- папки с обновленным файлом.

Именно на этом этапе вы можете экспериментировать с несколькими файлами, поскольку программа предлагает вам ввести имя папок, а не файлов.

Поскольку мы собираемся создать заплату только для одного файла, сбросьте флажок Include Sub-Directories (Включить подпапки).

3. Установка названия программы.

Здесь можно выбрать язык для программы заплаты (французский или немецкий).

Вы можете также присвоить имя программе заплаты. Это имя будет представлено конечному пользователю в главной текстовой области (но не в строке заголовка), когда он запустит программу заплаты. Следовательно, имя должно включать название обновляемого приложения и номер новой версии.

Если вы не очень ясно представляете себе, как будет выглядеть программа заплаты после запуска ее пользователем, можете щелкнуть на кнопке Preview (Предварительный просмотр), чтобы взглянуть на экран, который увидит конечный пользователь.

4. Вставка дополнительной информации.

Этот экран позволяет ввести дополнительный текст (как для файла Readme) о выполненных обновлениях. Здесь нет кнопки, позволяющей задать в качестве источника такой информации уже подготовленный файл, но определенной автоматизации можно достичь путем ввода текста в заданное поле методом копирования и вставки. После добавления необходимой с вашей точки зрения информации я советую просмотреть весь вставленный текст, чтобы убедиться в корректном его форматировании. Для этого щелкните на кнопке Preview.

5. Установка окна заплаты.

Теперь можно выбрать желаемый размер экрана инсталляции и главный заголовок, который увидит пользователь, запустив программу инсталляции.

6. Определение и использование растровых изображений для программы Patch Maker.

В этой части мастер позволяет вам разместить в программе инсталляции два растровых изображения. Однако не забывайте, что дополнительные файлы увеличивают общий размер заплаты.

Первое изображение должно иметь строго определенный размер — 128 пикселей в ширину и 280 пикселей в высоту при использовании 256 цветов. Это изображение будет выведено на главном ознакомительном экране и слева от текста в главном текстовом поле. Чтобы лучше представить себе результат включения изображения, выберите любую картинку и щелкните на кнопке Preview.

Второе изображение будет использовано в качестве фона главного экрана и должно включать не более 256 цветов. Проверьте свой выбор с помощью кнопки Preview.

7. Установка папки инсталляции, предлагаемой по умолчанию.

Этот экран позволяет ввести имя папки, в которую будет инсталлировано приложение и связанные с ним файлы. Вот пример.

```
"C:\Program Files\MyCompany\MyApplcation"
```

Стандартная папка заплаты должна совпадать с папкой, используемой в программе инсталляции. Если вы направите заплату не в ту папку, она не справится со своей задачей и пользователю, который попытается ее реализовать, будет возвращено сооб-

щение об ошибке “Nothing to do” (Ничего не выполнено). Пользователь также должен знать о необходимости соблюдения этих условий работы. Если он запустит на выполнение заплату не в той папке, где инсталлировано приложение, работа заплаты будет обречена на неудачу.

Кроме того, в этом разделе мастера вам предлагается возможность считать папку инсталляции из `*.ini`-файла, который находится в данный момент на принимающем компьютере, или из записи системного реестра принимающего компьютера.

Если вы щелкнете на кнопке Registry base, то получите подсказку трех значений элементов системного реестра: Root Key (Корневой ключ), `Key` (Ключ) и `Sub Key` (Подключ). Эти значения должны быть установлены вами при разработке.

8. Завершающий экран.

На последнем экране мастера программа Patch Maker предоставляет возможность создать, наконец, программу обновления или отказаться от своего намерения. Если вы уверены в корректности установки всех параметров, смело щелкните на кнопке Finish (Готово), в результате чего будет запущен процесс создания программы заплаты. Если вы в чем-то не уверены, щелкните на кнопке Back (Назад), чтобы перепроверить параметры программы.

9. Построение программы заплаты.

После того как вы, отбросив сомнения, примете окончательное решение строить программу заплаты, вам будет предложено указать, где создавать `.exe`-файл и как его назвать. Отвечая на этот запрос мастера, имейте в виду, что программа заплаты не должна располагаться в одной папке (или подпапке) с файлами основного приложения.

Когда долгожданный файл будет построен, можете сохранить файлы проекта в безопасном месте (в полной боевой готовности для будущих обновлений и исправлений, если таковые придется вносить).

Всегда стоит протестировать новоиспеченную программу заплаты. Лучше всего для этого открыть окно утилиты Windows Explorer и запустить файл (так бы поступил пользователь, ради которого вы и старались).

Убедитесь, что вы инсталлировали файл приложения, который должен быть обновлен, в папке, заданной по умолчанию. Запустите программу инсталляции, а затем откройте файл приложения, чтобы убедиться во внесении нужных изменений. Если обновление заключалось в исправлении ошибки, которая не видна “невооруженным глазом”, то лучший метод проверки успешной работы заплаты — проверить номер версии файла приложения.

Несколько советов, касающихся обновлений и заплат

Эти советы помогут вам решить, когда и как следует обновлять приложение.

- Когда обновление станет доступным, максимально широко проинформируйте об этом пользователей — по электронной почте и через свою Web-страницу.
- Не следует навязчиво бомбардировать пользователей информацией об обновлении — это может иметь обратный результат.
- Обнаруженные ошибки нужно исправлять как можно быстрее, и так же без промедления следует сделать такого рода обновление доступным для пользователей.

- Никогда не отправляйте обновленную версию или заплату по электронной почте, не получив сначала от пользователя разрешение на это.
- Если вы внесли в свое приложение небольшие изменения, не стоит на их базе создавать новую версию. Вероятно, целесообразнее подождать внесения более серьезных изменений.

Управление версиями и программа TeamSource

Общеизвестно, что транзакции разрушают старые данные.

Любую версию данных можно рассматривать как "снимок" данных в конкретный момент времени. Так вот, под управлением версиями понимается не что иное как хранение резервных копий этих снимков. На "диалекте" управления версиями создание такого снимка означает "сдавать" данные в "архив".

Самые ценные для программиста данные — исходный код, который постоянно изменяется. Иногда изменения заключаются в добавлении новых фрагментов программы, но чаще всего модифицируется существующий код. При каждом изменении строки программы ее старая версия теряется. Использование такой программы управления версиями, как TeamSource, позволяет сохранять резервные копии старых версий программы.

TeamSource — это отдельная программа, поставляемая в комплекте с C++Builder (вариант Enterprise Edition) и Delphi. Она также распространяется как независимый программный продукт. TeamSource используется для отслеживания изменений, которым подвергается исходный код в процессе эволюции проекта.

Программа TeamSource особенно полезна в сочетании с проектами C++Builder (или Delphi), причем ее возможности не ограничиваются только этими проектами. Она полезна в работе с любым проектом, в котором задействовано множество файлов, хранимых в одной папке (и, возможно, в подпапках). Если в проекте существуют текстовые файлы особого значения, TeamSource может обеспечить дополнительную обработку этих файлов, но лучше всего она справляется с управлением версиями для набора спецификаций (поддерживаемых в виде документов Word) или набора Web-страниц (в виде HTML-документов и файлов изображений) и пр.

Кто должен использовать программу TeamSource

Программа TeamSource предназначена для использования в команде разработчиков (это подчеркнуто в названии: *team* — по-английски команда), но она также полезна и для тех, кто работает в одиночку.

Почему следует использовать программу TeamSource

Программа TeamSource позволяет одному разработчику поддерживать список всех изменений проекта. Когда оценка прошедших событий показывает, что на пути эволюции проекта был сделан неправильный "поворот", TeamSource позволит вернуться назад. Кроме того, эта программа позволяет более уверенно продвигаться вперед, поскольку вы знаете, что любой неверный шаг можно с ее помощью легко скорректировать.

Назад следует возвращаться, если становится очевидным, что последнее нововведение оказалось столь неработоспособным или неэффективным, что должно быть отменено. Одна-

ко чаще всего требуется заглянуть в старые версии, чтобы взглянуть на исходный код в том виде, как он передавался пользователям. Хотя код с тех пор мог сильно измениться, пользователи именно той версии могут предъявлять вам претензии.

TeamSource может отслеживать одновременно несколько версий одного проекта. Команда может разрабатывать один проект в различных направлениях. Например, приложение, работающее с базой данных служащих, нужно обновить по двум пунктам: перевести на "рельсы" модели клиент/сервер и в саму базу данных добавить несколько таблиц. Без программы TeamSource эти изменения следовало бы обрабатывать последовательно. С использованием же TeamSource можно параллельно вести разработку в двух направлениях, объединив их на заключительной стадии.

Программный проект — это сложный объект, который может состоять из различных частей, а каждая часть может быть подобъектом разных подсистем управления версиями — более стабильных библиотек, которые изменяются менее часто, и кода проекта, который может изменяться чуть ли не ежедневно.

Когда использовать TeamSource

Использование программы TeamSource должно опираться на обоснованную стратегию. Самая простая стратегия заключается в том, что свои файлы следует архивировать в конце каждого рабочего дня. Введение новшества требует обычно нескольких дней, поэтому следует зафиксировать версии проекта в начале и конце этого периода, чтобы при необходимости можно было вернуться назад к нужным вехам (закладкам).

Более целесообразная стратегия состоит в архивации файлов для каждой транзакции. В разрабатываемом коде под транзакцией понимают все модификации, внесенные для расширения или изменения характеристик кода. В понятие характеристики входит и количество переменных, и любая значительная деталь, которую часто можно разбить на задачи, подзадачи, подзадачи подзадач и т.д. С другой стороны, такого рода детали обычно заранее четко определяются в большинстве команд программистов.

Где использовать TeamSource

Программу TeamSource можно использовать в локальной сети (LAN) или на отдельном компьютере. Такая программа управления версиями, как TeamSource, — это дополнительная форма резервирования исходного кода. Поэтому архивы следует хранить на отдельном от рабочего проекта устройстве, что и позволяет реализовать программа TeamSource.

Если для определения сроков выполнения архивирования используется разумная стратегия, TeamSource может оказаться лучшей формой резервирования по сравнению с копированием файлов в ночную смену. При возникновении серьезных неприятностей вы скорей всего захотите вернуться к моменту "до добавления этой формы", чем просто к состоянию на "прошлый вторник".

Случаются и катастрофы, и тогда необходимо вернуться к максимально возможному недавнему состоянию, поэтому программу TeamSource можно рассматривать не как замену обычного резервирования, а как его усовершенствование.

Как использовать TeamSource

В основе программы TeamSource, как и пакета C++Builder, лежит принцип использования проектов, а в основе проектов — понятие папки.

TeamSource более жестко связана с папками, чем C++Builder. Программа TeamSource "исходит" из того, что проект состоит из папки и подпапок, при этом никакие другие файлы в

расчет не принимаются. Это не отражается на гибкости, допустимой в среде C++Builder. Ведь в C++Builder проект могут составлять файлы, которые одновременно участвуют в другом проекте и находятся в отдельной папке. Поскольку программа TeamSource обладает меньшей гибкостью в этом смысле, то перед ее использованием часто имеет смысл реорганизовать папки в соответствии с ее требованиями.

Новые проекты

Программа TeamSource требует большего внимания к проекту, чем обычно. Однако усилия, затраченные на подготовку к работе с этой программой окупятся сторицей, т.е. вы можете быть уверены в том, что ее дальнейшее использование в нормальном режиме не вызовет никаких трудностей и принесет желаемый эффект, и вы будете использовать ее регулярно.

При первом использовании программы TeamSource вам будет предложено ввести свое имя и адрес электронной почты, а затем вам придется создать новый проект. Для этого из меню File выберите команду New Project (Создать проект). Откроется диалоговое окно, которое предоставит возможность выбрать между импортированием существующего проекта и созданием нового "с нуля".

Импортирование нового проекта

Самостоятельный разработчик обычно создает проекты "с нуля". При групповой работе один член команды создает проект, а другие позднее импортируют его. Импортировать проект совсем несложно — достаточно использовать стандартное диалоговое окно открытия файла, чтобы найти созданный ранее файл проекта и указать папку на локальном компьютере, в которой следует хранить этот проект (см. раздел "Задание локальной папки" ниже в этой главе). Все файлы проекта затем копируются в эту папку, а проект будет обновляться на основе более новых версий, обнаруженных в локальной папке.

Создание нового проекта „с нуля“

Создание нового проекта предусматривает ряд этапов.

- Создание проекта.
- Определение локальной папки.
- Задание параметров проекта.
- Предоставление возможности другим пользователям импортировать проект.

Создание проекта

Создание проекта "с нуля" можно сравнить с работой под "руководством" мастера — вам нужно "пройти" семь диалоговых окон, которые потребуют введения данных, касающихся проекта. (Не пугайтесь размеров списка — значения, которые программа TeamSource предлагает использовать по умолчанию, обычно вполне подходят для пунктов этого списка.)

1. The Project You Are Creating (Создаваемый проект). Существует три основных вида данных, которые необходимо указать для проекта.
   - **1.1.** Имя проекта. Введите описательное имя на естественном языке (пробелы разрешены).
   - **1.2.** Имя файла проекта. По умолчанию предлагается имя проекта, но с удаленными пробелами. Вы можете сократить это имя по своему усмотрению. Не включайте в состав имени файла данные о пути; путь будет указан ниже.

1.3. Средство управления версиями (контроллер версий). По умолчанию предлагается Zlib (Borland), хотя возможны и другие варианты.

2. The Project Folder (Папка проекта). Здесь указывается адрес (путь) папки, в которой будут храниться файлы проекта, курируемого программой TeamSource. Укажите путь либо непосредственным вводом, либо с помощью просмотра каталогового дерева, доступного после щелчка на кнопке с тремя точками (…). Обычно эту папку размещают на отдельном (от вашего C++Builder-проекта) компьютере или по крайней мере на отдельном диске. При совместном использовании TeamSource-проектов несколькими пользователями по локальной сети эту папку обычно создают на сервере.

Необходимо обеспечить для всех, кто работает с этим проектом, достаточные права доступа к этой папке.

3. The Project Subfolders (Подпапки проекта). TeamSource использует папку `Archive` (запись версий поддерживаемого кода), папку `History` (которая содержит список архивных составляющих для проекта) и еще одну для файлов блокировки (которые позволяют одному пользователю выполнять запись данных в проект, не допуская к этому других). Эти три папки по умолчанию являются подпапками главной папки проекта, но при желании вы можете их изменить.
4. The Mirror Tree (Зеркальное дерево). TeamSource может поддерживать зеркальное отображение проекта (копию текущего состояния всех файлов проекта). Если вы желаете иметь такое отображение, установите флажок опции Enable Mirror Tree (Разрешить зеркальное дерево) и укажите папку, в которой оно будет храниться. В качестве резервного логического устройства по умолчанию предлагается подпапка основной папки проекта, но вы можете задать отдельный компьютер или диск.
5. Publishing (Публикация). Вы можете указать итоговый файл, который содержат комментарии, сделанные при архивации разработчиками группы файлов. Можно также задать системный журнал, который будет включать больше информации о сдаваемых в архив файлах (например, кто и когда их сдает, состав файлов, а также общее резюме по группе архивируемых файлов и пофайловые комментарии). Кроме того, можно указать SMTP-сервер (Simple Mail Transfer Protocol — простой протокол электронной почты), чтобы позволить программе TeamSource послать по электронной почте уведомления о новых поступлениях в архив.
6. Email Addresses (Адреса электронной почты). Если вы указали SMTP-сервер на предыдущем этапе, то теперь следует ввести адреса электронной почты всех тех, кого необходимо уведомлять об изменениях в проекте. После любого изменения каждому адресату будет послано уведомление, содержащее следующие данные:
   - итоговый файл (см. предыдущий этап);
   - системный журнал (см. предыдущий этап);
   - список модифицированных файлов.
7. Confirmation (Подтверждение). На последнем этапе отображается вся введенная ранее информация, позволяя вам либо убедиться в ее корректности, либо вернуться к предыдущим этапам для изменения каких-либо данных.

Из всех перечисленных этапов самыми значительными являются следующие:

- задание имени проекта;

- задание папки проекта;
- задание SMTP-сервера;
- задание адресов электронной почты для уведомлений.

Введя эти четыре элемента, дальше можно (как правило) просто щелкать на кнопке Next и позволить программе TeamSource заполнить остальные элементы. Для одного разработчика достаточно лишь первых двух элементов — имя проекта и папка для проекта.

Задание локальной папки

После создания проекта программа TeamSource предложит выбрать локальную папку, в которой будет храниться ваш C++Builder-проект.

Обычно в этом случае выбирают ответ Yes, но возможен и выбор ответа No, если C++Builder-проект находится не на этом компьютере (см. раздел "Обработка локальных папок" ниже в этой главе) или по другим причинам.

Программа TeamSource затем предложит диалоговое окно, позволяющее добавить локальные папки. Введите адрес локальной папки или найдите его с помощью кнопки с многоточием ( ). Можно ввести даже несколько папок, и тогда в каждой будет сохранен полный архив.

Затем программа TeamSource предложит запустить Content Wizard (мастер проверки содержимого) по локальным папкам. Можно, конечно, отложить это на "потом", но обычно соглашаются, отвечая выбором варианта Yes. Этот мастер просмотрит содержимое локальных папок, а затем отобразит диалоговое окно с этими папками и их подпапками. Вы можете щелкать на каждой папке по очереди, чтобы просмотреть список содержащихся в ней типов файлов.

Программа TeamSource поддерживает файлы всех перечисленных типов. Возможно, вам захочется отредактировать эти списки, чтобы, например, удалить промежуточные файлы (с расширением `*.obj` и `*.il*`), резервные файлы (`*.~cpp`) и другие "обломки", которые имеют тенденцию накапливаться в таких папках (`*.tmp`, `*.cgl` и пр.).

Разобравшись с проектом (локальным и удаленным, т.е. архивным его вариантом), вы возвратитесь к главному экрану программы TeamSource, на котором будет отображена итоговая информация о проекте.

Параметры проекта

Пользователь, который создал проект, имеет к нему доступ администратора (Administrator). Прежде чем с проектом можно будет работать, пользователю стоит просмотреть параметры, определяющие характер его использования. Такой подход можно применить в случае одиночного пользователя, для работы же в команде он просто необходим, поскольку позволяет установить, какие пользователи будут иметь доступ к проекту.

Чтобы установить нужные параметры, вам следует получить блокировку уровня администратора (Administrator Lock) (см. раздел "Запрос на блокировку" ниже в этой главе), а затем выбрать из меню Project команду Options (Параметры). В результате откроется диалоговое окно с четырьмя вкладками: General (Общие), Directories (Папки), Users (Пользователи) и Publishing (Публикация).

Вкладка General

Вкладка General диалогового окна Project Options (Параметры проекта) позволяет сделать следующее.

- *Изменить имя файла, содержащего информацию о версии*. TeamSource может поддерживать файл, который содержит номер текущей версии проекта. Этот файл будет храниться в корневой папке архива.

- *Обнаружить новые локальные папки*. При выборе этой опции программа TeamSource будет пытаться найти новые подпапки на локальном компьютере. Если у вас есть подпапки, которые относятся к C++Builder-проекту, но вы не хотите, чтобы TeamSource поддерживала их версии, сбросьте флажок этой опции.
- *Установить необходимый уровень комментариев при записи файлов в архив*. Можно выбрать вариант Summary Comments (по одному комментарию на группу файлов) или File Comments (отдельные комментарии для каждого файла в группе). При выборе любого флажка разработчики не смогут сдать свои файлы в архив без комментариев соответствующего уровня.
- *Сообщить программе TeamSource начальный номер версии проекта*. Эта опция особенно полезна при создании нового архива TeamSource для старого C++Builder-проекта с существующим номером версии, уже поддерживаемым средствами C++Builder. Формат номера версии одинаков для обеих программ (например, полный номер версии имеет вид 1.3.23.8, где 1 — основной номер версии, 3 — дополнительный номер версии, 23 — промежуточный номер и 8 — номер создаваемого проекта).

Вкладка Directories

Вкладка Directories позволяет изменять папки, в которых программа TeamSource хранит архив. Эти папки совпадают с папками, использованными для создания проекта (см. пп. 3 и 4 раздела "Создание проекта" выше в этой главе).

Вкладка Users

Во вкладке Users можно указать уровень доступа пользователей к проекту. Можно ввести любое желаемое количество пользователей, определив для каждого из них уровень доступа.

- Read-Only (Только для чтения). Такой пользователь может копировать файлы из архива, но не имеет права изменять его. В частности, такой пользователь не может записывать свои файлы в архив.
- Read-Write (Для чтения и записи). Такой пользователь может записывать свои файлы в архив.
- Administrator (Администратор). Если вы предоставили пользователю уровень доступа Read-Write, можно также наделить его правом администратора. Это дает ему возможность записывать в архив файлы и изменять параметры проекта (например, такой пользователь может использовать эту вкладку диалогового окна для введения новых пользователей).

После добавления пользователей их можно удалять или "редактировать". "Редактирование" пользователя означает изменение его имени или определение ему другого уровня доступа. Однако пользователь не может редактировать данные о себе самом.

Имя пользователя, заданное в диалоге с программой TeamSource, совпадает с зарегистрированным именем пользователя в системе Windows.

В нижней части этой вкладки предусмотрен флажок опции Allow Guest Access to This Project (Позволить доступ к этому проекту на правах гостя). После установки этого флажка любой пользователь может получить доступ к проекту, причем этот доступ будет ограничен уровнем "только для чтения".

Если вы работаете в среде Windows 9x и при этом не используете конкретное пользовательское имя, то вы будете зарегистрированы под именем "Guest". В этом случае пользователь `Guest` по отношению к проекту будет обладать полными правами доступа администратора.

Вкладка Publishing

Вкладка Publishing позволяет внести изменения в способ уведомления пользователей по электронной почте о поступлениях в архив. Состав данных здесь такой же, как при создании проекта (см. пп. 5 и 6 раздела "Создание проекта" выше в этой главе).

Обработка локальных папок

С "точки зрения" программы TeamSource локальной считается папка, в которой хранится C++Builder-проект. В менее простых случаях это хранилище буквально нельзя назвать локальной папкой по причине отсутствия ее на вашем компьютере или же использования не одной, а нескольких папок.

Локальная папка может не принадлежать вашему компьютеру, поскольку вы просто настраиваете TeamSource-проект, который будут использовать другие разработчики. Такая ситуация возникает в том случае, когда руководитель группы устанавливает архив таким образом, чтобы иметь возможность получать к нему доступ на правах администратора, а другие разработчики могли поддерживать C++Builder-проекты на других компьютерах с правами записи файлов в архив.

В такой ситуации руководитель группы может посчитать нецелесообразным иметь копию C++Builder-проекта на своем компьютере и поэтому не станет задавать в качестве местоположения локальной папки локальный компьютер.

Локальная папка может представлять собой не одну папку, а несколько. TeamSource прекрасно справляется с такой ситуацией, если все эти папки имеют общий корень. Например, если C++Builder-проект содержит файлы `D:\programming\projects\mailApp\form1.cpp` и `D:\programming\libraries\internet\my-email.cpp`, было бы разумно в качестве локальной папки установить папку `D:\programming\`.

В таком сценарии не исключено существование других подпапок папки `D:\programming\`, и их следует удалить из каталогового дерева архива. Кроме того, вполне вероятно, что в папке `D:\programming\libraries\internet\` могут находиться файлы, не имеющие отношения к архиву `mailApp`. Первоначально все файлы этой папки должны быть включены в архив, а впоследствии некоторые из них можно по отдельности исключить.

В качестве альтернативной стратегии для этого примера можно рассматривать поддержку двух отдельных архивов: одного — для проекта `mailApp`, а другого — для проекта `library\internet`. В этом случае упрощается создание архивов, но усложняется их поддержка. После работы над проектом `mailApp` разработчик вряд ли забудет обновить архив `mailApp`. Если же во время работы обнаружится небольшая ошибка в файле `my-email.cpp`, которая тут же была исправлена, то о необходимости обновить отдельный архив можно легко забыть.

Программа TeamSource позволяет определить несколько "локальных папок", и в каждой из них будет поддерживаться полная копия архива. Если ваш проект C++Builder включает несколько папок, вам не нужно указывать их в качестве локальных.

Окна программы TeamSource

В программе TeamSource предусмотрено три различных способа просмотра файлов в архиве, и для каждого способа существует свое окно. TeamSource определяет эти окна как представления ("views"), поэтому это слово присутствует в названиях окон: Remote View (Удаленное, или архивное представление), Local View (Локальное представление) и History View (Представление в "историческом" ракурсе).

Окно Remote View служит для отображения файлов по мере их поступления в центральный архив. Окно Local View предназначено для отображения изменений, которые пришлось внести в файлы на локальном компьютере, чтобы синхронизировать их с центральным архивом. Окно History View предоставляет возможность просмотреть журнал учета изменений, внесенных в архив.

Окно Remote View

Чтобы открыть окно "с видом" на удаленную папку (папки) архива, можно либо нажать клавишу <F7>, либо щелкнуть на расположенной слева пиктограмме Remote, либо выбрать из главного меню окна программы TeamSource команду View⇨Remote Project (Представление⇨Архивный проект). Это окно разделено на две панели. Слева (в стиле программы Windows Explorer) отображается дерево папок, а справа — списочное представление версий файлов.

Свойства архивных (нелокальных) папок

При щелчке правой кнопкой мыши на удаленной (т.е. архивной) папке открывается меню с командами, позволяющими просмотреть ее свойства. Если до щелчка правой кнопкой мыши вы получите блокировку уровня администратора, значит, у вас есть право добавлять или удалять подпапки (команды Add или Delete соответственно) и редактировать свойства (команда Properties). Свойства папок содержат информацию о средстве управления версиями, составе включаемых и удаляемых файлов, а также *порождающие правила* (production rules). Например, такое порождающее правило, как `.cpp -> .obj`, информирует программу TeamSource о том, что файлы с расширением `.obj` создаются на основе `.cpp`-файлов с таким же именем). Следовательно, программа TeamSource не обязана хранить версию `.obj`-файла, если существует `.cpp`-файл с таким же именем.

К сожалению, такие правила должны установиться индивидуально для каждой папки каждого проекта. Невозможно установить общие порождающие правила для всех проектов.

Действия архивных (нелокальных) файлов

При щелчке правой кнопкой мыши на архивном файле открывается меню с рядом возможных действий (они перечислены ниже). Из них чаще всего используется действие Compare Revisions (Сравнить исправления).

- View Tip Revision (Просмотреть последнюю версию). Обеспечивает отображение самой последней версии файла.
- View Any Revision (Просмотреть любую версию). Позволяет выбрать версию файла и просмотреть ее.
- Save Revision As (Сохранить версию как). Позволяет выбрать версию файла, а затем сохранить ее на диске.
- Delete from Project (Удалить из проекта). Перед удалением всех версий файла из проекта необходимо заблокировать проект.
- View Archive Report (Просмотреть архивные данные). Позволяет узнать, когда версия файла была сдана в архив, кем и пр.
- Compare Revisions (Сравнить исправления). Открывает диалоговое окно, позволяющее задать две версии файла, после чего программа TeamSource отобразит результаты сравнения этих двух версий, выделив строки, которые были добавлены или удалены из одной версии для образования другой. При сравнении игнорируются различия между файлами (неодинаковое количество пробельных символов), если задать такой вариант сравнения с помощью команды Options⇨Preferences (Параметры⇨Предпочтения).

Окно Local View

Чтобы получить локальное представление об архиве, можно либо нажать клавишу <F8>, либо щелкнуть на расположенной слева пиктограмме Local, либо выбрать из главного меню окна программы TeamSource команду View⇨Local Project (Представление⇨Локальный проект). Это окно разделено на две панели: слева отображаются локальные изменения, а справа — архивные.

Поскольку в этом представлении отображается архив с "точки зрения" локального компьютера, именно оно и используется разработчиками чаще всего. Файлы, которые чаще всего изменялись разработчиком, находятся на локальном компьютере, поэтому при их просмотре следует использовать возможности локального представления. Файлы, добавленные другими членами команды разработчиков, отсутствуют на локальном компьютере, но могут быть скопированы туда с помощью локального представления.

Для каждого файла, показанного в локальном представлении, программа TeamSource рекомендует некоторое действие, позволяющее сохранять архив "на плаву". Например, если разработчик изменил файл на локальном компьютере, этот файл будет отображен в окне Local View с рекомендациями по внесению его в архив.

Использование локального представления для записи файлов в архив

При повседневном использовании программы TeamSource самым популярным действием является запись файлов в архив. В отличие от других средств управления версиями, перед началом работы файлы не нужно блокировать. Разработчики могут изменять их локальные копии, а затем записывать в архив модифицированные файлы.

Обычно разработчики выбирают локальное представление, чтобы просмотреть на правой панели список файлов, которые необходимо поместить в архив. Разработчик должен получить разрешение на блокировку, выбрать файлы и щелкнуть на кнопке Do It! (Выполнить) — выбранные файлы будут помещены в архив. Если проект требует ввода итоговых комментариев, это можно сделать после щелчка на кнопке Comment (Комментарии) до щелчка на кнопке Do It!.

Рекомендованные действия для файлов, отображаемых во вкладке Local View

Программа TeamSource, чтобы сохранить текущий архив, может дать ряд рекомендаций по внесению изменений в архивные или локальные файлы. Если действие рекомендуется для локальной версии файла, этот файл будет указан в правой панели. Если же действие рекомендуется для архивной версии файла, этот файл будет указан в левой панели. Если для файла не нужно выполнять никаких действий, такой файл не будет упоминаться в локальном представлении.

В левой и правой панелях эти рекомендации перечисляются в столбце Action (Действие) и обозначаются маленькими значками рядом с именами файлов. Эти рекомендованные действия базируются на сравнении данных о дате и времени создания локальных, архивных и базовых версий файлов.

Локальные версии — это файлы, хранимые на локальном компьютере, в то время как *архивные версии* хранятся в TeamSource-архиве. *Базовыми* называются версии, которые соответствуют времени синхронизации локальных и архивных файлов (последняя запись в архив, последнее извлечение и т.п.). При помещении файла в архив программа TeamSource записывает в файл `something.tsl` (если проект называется `something`) локальной папки следующую информацию: имя файла, номер версии и данные о времени создания.

В зависимости от различий между этими версиями, программа TeamSource может рекомендовать ряд следующих действий.

- Copy (Копировать). Архивная версия — более новая, и вам следует обновить собственную версию файла путем копирования файла из удаленного (архивного) на локаль-

ный компьютер. При выборе этого действия программа TeamSource перед выполнением перезаписи локального файла запросит подтверждение.

- Merge (Объединить). Архивная версия — более новая по сравнению с последней помещенной в архив, и вы к тому же изменили локальный файл с момента последней записи в архив. Средство управления версиями, которое обычно используется вместе с программой TeamSource (ZLib компании Borland), не поддерживает автоматического объединения, поэтому вам придется сделать это вручную. Если вы воспользуетесь средством управления версиями, которое поддерживает автоматическое объединение (например, PVCS), это будет сделано за вас.
- Merge by Hand (Объединить вручную). Автоматическое объединение выполнить нельзя, поэтому это нужно проделать вручную.
- Correct by Hand (Скорректировать вручную). Что-то не заладилось. В большинстве обычных случаев на основании результатов сравнения локальной, архивной и базовой версий программа TeamSource может определить, что следует сделать с файлом, но ей не всегда удается рекомендовать корректное действие. Например, если локальный файл датирован вторником, а базовая и архивная версии — четвергом, TeamSource в этой нестандартной ситуации не в состоянии рекомендовать какое-нибудь действие, поскольку локальный файл оказался более ранним, чем он был в момент последней сдачи в архив.

  Когда программа TeamSource рекомендует применить корректировку вручную, необходимо щелкнуть правой кнопкой мыши на имени файла и из контекстного меню выбрать команду View File Info (Просмотреть информацию о файле). Это позволит вам увидеть данные о времени создания локальной, архивной и базовой версий, которые известны программе TeamSource и которые помогут вам найти источник проблемы.

  Ошибки, которые вам предлагается скорректировать вручную (Correct by Hand), обычно вызываются факторами, "лежащими" вне сферы действия программы TeamSource (например, ситуация, описанная в предыдущем примере, могла быть вызвана простым перемещением старой копии файла в локальную папку). Кроме того (особенно при большом количестве таких ошибок), следует уточнить, как именно обрабатываются временные данные компьютерами, входящими в локальную сеть. У коллеги, с которым я совместно использовал проект TeamSource, большинство файлов сопровождалось пожеланием "скорректировать вручную", в то время как у меня встретилось лишь несколько таких рекомендаций. Ошибка же гнездилась на моем компьютере, часовой пояс которого был установлен равным Eastern Standard Time (восточноевропейское стандартное время), в то время как другие компьютеры корректно были настроены на вариант GMT (Greenwich Mean Time — среднее время по Гринвичу).
- Delete (Удалить локально). Файл был удален из проекта, поэтому его следует удалить с локального компьютера.
- Remove (Удалить из проекта). Файл был удален с локального компьютера, поэтому его также следует удалить из проекта.
- Touch (Синхронизировать). Дата локального файла была изменена, но содержимое осталось неизменным. При выборе этого действия выполняется синхронизация дат версий.

Использование локального представления для выполнения других действий

Используя контекстное меню или клавиши ускоренного доступа ("горячие" клавиши), с перечисленными во вкладке Local View файлами можно выполнить ряд действий.

- View Local Changes (Просмотреть "локальные" изменения). Отображает различия между локальным файлом и самой последней версией, взятой из архива (базовой версией).

- View All Changes (Просмотреть все изменения). Отображает различия между локальным файлом и самой последней версией в проекте (версией TeamSource-архива).
- View Remote Changes (Просмотреть архивные изменения). Отображает различия между архивной и базовой версиями.
- View File Info (Просмотреть информацию о файле). Отображает даты создания локальной, базовой и архивной версий.
- Revert (Вернуть прежний файл). Копирует архивный файл в локальный файл.
- Change the Comments for the Selected File (Изменить комментарии к выбранному файлу). Позволяет просмотреть и изменить комментарии к файлу.
- Change the Recommended Action for the File (Изменить для файла рекомендованное действие). Позволяет выбрать другое действие для файла. Обычно эта команда используется для изменения рекомендованного действия с варианта Merge by Hand либо на вариант Copy либо на вариант Check-In.
- Move Files from One Pane to the Other (Переместить файлы с одной панели на другую). Выполнение некоторых действий возможно только на одной панели, поэтому не исключено, что для выполнения нужного действия вам придется переместить какой-нибудь файл на другую панель.

Вкладка History View

Эта вкладка также состоит из двух панелей. Слева отображается список операций по записи файлов в архив с указанием даты и исполнителя. Справа — информация о выбранной группе файлов, итоговые комментарии и список файлов, входящих в эту группу.

Средства управления версиями

Программа TeamSource по умолчанию использует в качестве средства управления версиями программу ZLib (компания Borland), а также обеспечивает поддержку программы PVCS (компания Merant), хотя она может поддерживать любое средство такого плана. Задать (или изменить) используемое средство можно с помощью меню проекта Controllers (Средства управления версиями).

Если вы хотите использовать другое средство управления версиями, необходимо позаботиться о TeamSource-расширении (`.tsx`-файле). Это — DLL-библиотека, которая обеспечивает дополнительные возможности (или просто взаимодействует с другим средством управления версиями) путем реализации нужных интерфейсов.

Закладки

В “летописи” версий, архивируемых программой TeamSource, можно устанавливать закладки. Под *закладкой* понимается стадия, к которой можно легко вернуться, поскольку версию всех файлов проекта, которые существовали на момент создания конкретной закладки, можно легко восстановить.

Закладка может быть особенно полезной, когда соответствующая ей версия совпадает с выпущенной версией программного продукта. Допустим, что пользователь, который работает с версией, выпущенной много месяцев или даже лет назад, обнаружил проблему, для которой нужно найти решение. Если при выпуске той версии была создана закладка, вы можете быстро вернуться к соответствующей стадии проекта и локализовать проблему.

Чтобы установить закладку, необходимо выбрать из меню Project команду Bookmarks. Откроется диалоговое окно, которое позволяет вводить (добавлять), редактировать и удалять закладки. Если вы собираетесь добавить закладку, вам придется присвоить ей имя и решить ее "судьбу" — сделать ее локальной либо глобальной. Локальная закладка доступна только пользователю, который ее создал. Глобальная же — всем пользователям, но ее может создать лишь пользователь, который по отношению к проекту обладает правом доступа администратора.

Извлечение проектов

При обычном ходе событий локальный проект, по всей вероятности, соответствует более новому варианту, чем архив, хранимый на удаленном (дистанционно) компьютере, поскольку изменения вносятся на локальном компьютере, а позже модифицированные файлы попадают в архив удаленного компьютера.

Однако если в проекте занято несколько разработчиков или на локальном компьютере обнаружились потери (например, в результате искажения данных), тогда файлы архива могут оказаться более свежими. В таком случае вам, возможно, захочется "вытащить" проект, т.е. скопировать все файлы из отдаленного архива в локальный компьютер.

Это можно сделать с помощью команды Pull To (Извлечь) из меню Project. Существует также вариант быстрого извлечения (с помощью команды Fast Pull), когда с удаленного компьютера вытаскиваются только файлы, созданные в последнее время. При выполнении команды обычного (не быстрого) извлечения с удаленного на локальный компьютер копируются все файлы.

Еще один вариант извлечения проекта выполняется по заранее созданной закладке. В этом случае файлы локального компьютера могут перезаписаться "более древними" архивными файлами, поэтому перед осуществлением такой операции стоит сначала зарезервировать локальные файлы (поместив их в архив).

Блокировки

В блокировках возникает необходимость, когда один пользователь собирается изменить архив. Самой распространенной причиной такого вмешательства является желание поместить в него файлы. Кроме того, после создания проекта обычно запрашивают блокировку, чтобы изменить параметры проекта.

Блокировка с помощью TeamSource, в отличие от других систем управления версиями, заключается в том, что вы можете блокировать только целый архив, а не отдельные файлы. Более того, используя программу TeamSource, вам нужно накладывать блокировку только на поступающий в архив файл, в отличие от других систем управления версиями, которые могут потребовать извлечения файла, связанного с блокировкой, сохранения блокировки на время внесения изменений, а затем возвращения этого файла, благодаря чему блокировка будет снята.

Запрос на блокировку

Сделать запрос на блокировку можно из меню Project или нажатием клавиши <F4>. В любом случае вам будет предложено сообщить о том, как долго вы собираетесь сохранять блокировку, причем по умолчанию предлагается пятиминутный период. В конце этого периода вы не потеряете блокировку, но затем запросить блокировку смогут другие пользователи. Если у вас есть доступ к проекту с правом администратора, вы можете также запросить блокировку уровня администратора, которую никто не сможет оспорить и которая может сохраняться даже после закрытия программы TeamSource.

Кроме того, у вас есть возможность внести комментарий по поводу блокировки, чтобы объяснить другим пользователям, почему вы заблокировали проект.

Оспаривание блокировок

В нижней части главного окна программы TeamSource отображается список блокировок текущего проекта. Сюда входят предоставленные, запрашиваемые и продленные блокировки.

После щелчка правой кнопкой мыши на элементе списка отображается меню, которое позволяет снять блокировку, изменить разъясняющий ее комментарий или продлить время ее действия. Кроме того, это меню можно использовать для передачи разрешения на блокировку другому пользователю из очереди. Это меню содержит еще одну команду, именуемую Verify Current Lock (Проверить текущую блокировку). При выборе этой команды происходит "покушение" на текущую блокировку.

Когда запрос на блокировку происходит в пределах заявленного для нее времени, вам сообщат, сколько времени еще осталось "в распоряжении" у этой блокировки. По истечении времени пользователь, запросивший блокировку, будет уведомлен о "покушении" на нее. Для этого пользователя откроется диалоговое окно, которое он может использовать для снятия блокировки или для продления времени ее действия. Если пользователь не ответит на это диалоговое окно в течение двух минут, блокировка автоматически снимается.

Использование программы InstallShield Express

В этом разделе мы рассмотрим использование версии программы InstallShield, которая поставляется вместе с C++Builder 5. Мы покажем, как создать собственную программу инсталляции для готового приложения. При этом мы не берем на себя невыполнимую задачу рассмотреть сценарии создания программ инсталляции для всех известных типов приложений, но мы познакомим вас с основами построения таких программ. InstallShield предоставляет массу возможностей, которыми вы как разработчик можете воспользоваться. Закладываемый здесь фундамент подойдет для любого разрабатываемого приложения, дополнения же при необходимости вы легко сделаете сами.

Установка программы InstallShield

Программа InstallShield распространяется бесплатно в составе пакета C++Builder, начиная с версии C++Builder 1. Большинство из вас по крайней мере визуально знакомы с программой InstallShield, даже если вам еще не приходилось самим создавать программу инсталляции для нового приложения.

Если вы еще не установили программу InstallShield, вставьте компакт-диск с C++Builder 5 в дисковод для компакт-дисков и... автостарт не заставит вас ждать. Если (ну, всякое бывает) ничего не произойдет, перейдите в папку `Isxpress` компакт-диска и дважды щелкните на имени файла `Setupex.exe`. Тем самым вы запустите программу инсталляции для приложения InstallShield Express.

Версия программы InstallShield, которая поставляется в составе C++Builder, не является полной коммерческой версией; Это специальная версия, подготовленная для C++Builder, и в этом варианте некоторые дополнительные возможности полной версии недоступны. Однако в ней есть все, что может понадобиться каждому разработчику для распространения своих приложений в реальном мире.

Итак, начнем освоение InstallShield

После установки приложения InstallShield Express самое время подумать о типе инсталляции, который подойдет именно для вашего приложения. Например, установили ли вы при компиляции приложения в среде C++Builder такой параметр компоновщика, как Use Dynamic RTL (Использовать динамическую библиотеку этапа выполнения)? Если да, то придется ли вам подключать дополнительные `.dll`-файлы к вашему приложению? Кроме того, если вы выбрали опцию Build with Runtime Packages (Построить с пакетами времени выполнения) во вкладке Packages (Пакеты), то вам придется добавить дополнительные файлы в вашу программу инсталляции; в противном случае она не будет работать на другом компьютере. При выборе этих опций происходит жесткий вызов соответствующих файлов. Все дело в том, что если не выбрать эти опции, ваше приложение станет большим по размеру (я имею в виду `*.exe`- или `*.dll`-файл, с которым вы работаете), и наоборот. Но чтобы ваша программа могла работать на другом компьютере, необходимо включить некоторые дополнительные файлы.

Завершив разработку своего приложения, подумайте обо всех файлах, которые необходимы для его работы на другом компьютере. Сюда входят справочные файлы, файлы Readme, DLL-библиотеки и файлы данных (файлы базы данных и файлы примеров). Составив список всех необходимых файлов, лучше всего поместить их в отдельную папку, чтобы не "забыть" ничего в "экстремальных" условиях смены компьютера.

При первоначальном использовании приложения InstallShield вам будет предложено указать тип проекта инсталляции, который вы хотите создать. Щелкните на экране Create a New Setup Project (Создать новый проект установки). Затем присвойте своему проекту установки подходящее имя и выберите вариант создания этого проекта в новой папке. Это обеспечит сохранение файлов инсталляции и файлов проекта в одном легкодоступном месте.

Список Setup Checklist (рис. 29.4) — это та часть программы InstallShield, с которой вы будете работать после выбора опции Create a New Setup Project.

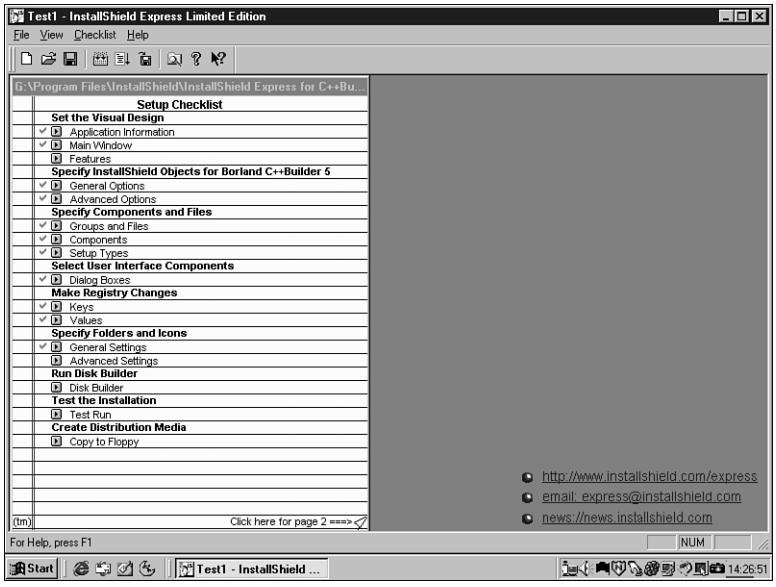


Рис. 29.4. Главный экран приложения InstallShield Express, позволяющий установить нужные параметры

Теперь пора познакомиться с главным экраном приложения InstallShield. Оно содержит множество опций, позволяющих создать программу инсталляции нужного вида. Эти опции представлены в форме списка, именуемого технологической картой (Setup Checklist), которую нужно внимательно просмотреть и соответствующим образом заполнить, т.е. установить параметры, используемые программой InstallShield.

Раздел Set the Visual Design (Установка визуальных характеристик)

Начните работу на главном экране с открытия диалогового окна Application Information (Информация о приложении), а затем введите имя программы, которое впоследствии будет представлено в диалоговом окне Add/Remove Programs (Установка и удаление программ).

Если вы выберете выполняемое приложение, InstallShield автоматически добавит заданный выполняемый файл в группу Program Files диалогового окна Groups and Files (Группы и файлы) и создаст пиктограмму для выполняемого файла в диалоговом окне General Icon Settings (Общие установки пиктограмм), если такого элемента еще нет в упомянутых диалоговых окнах.

Информация о версии должна быть сформирована на основании версии вашего приложения. Если вы установили номер версии в C++Builder, он автоматически будет размещен в соответствующем поле. Программа InstallShield использует этот номер версии для предотвращения перезаписи старых версий приложения новыми.

Соответствующим образом должно быть также установлено имя компании. Ваше приложение автоматически будет зарегистрировано в системном реестре, а все пути инсталляции и элементы стартового меню также получат значения по умолчанию.

Проверьте значение папки-приемника. Возможно, вам понадобится заменить вариант, предложенный программой InstallShield. В начале выражения пути должен остаться макрос `<Program Files>`. Папка вершины иерархического адреса на разных компьютерах может быть различной, поскольку не в каждом из них копия Windows инсталлирована на диске `C:\`, поэтому и конечным пользователям следует дать возможность выбора, где инсталлировать ваше приложение.

В разделе Main Window (Главное окно) введите текст и сообщите, какое графическое изображение вы желаете использовать в качестве фона во время инсталляции приложения. Это графическое изображение должно состоять только из 16 цветов. Если вы попытаетесь использовать 256 цветов, то получите от программы InstallShield сообщение об ошибке. Однако можно безнаказанно проигнорировать это сообщение и продолжать работу со своим проектом, если, конечно, вы уверены, что все конечные пользователи будут иметь оборудование, позволяющее отображать 256-цветные (и более) изображения. В противном случае изображение будет выглядеть искаженным.

В разделе Features (Возможности) рекомендуется оставить опцию Automatic Uninstaller (Автоматическое средство удаления приложения) установленной. Разъяснения по назначению этой функции выходят за рамки этой главы.

Раздел Specify InstallShield Options for Borland C++Builder 5 (Установка параметров InstallShield для Borland C++Builder 5)

В этом разделе выберите все элементы, которые применимы к вашему приложению в разделе General (Общие). Необходимость установки этих параметров определяется тем, придется ли добавлять для обеспечения работы приложения на другом компьютере дополнительные файлы.

При выборе опции BDE or SQL-Links (BDE или SQL-связи) вам придется включить и опцию BDE Control Panel File (Файл панели управления BDE). Если вы использовали пакеты времени выполнения, выберите из списка все, что может понадобиться для вашего приложения.

Если вы выберете опцию BDE Object (Объект BDE), то целесообразно использовать полную инсталляцию BDE. В противном случае возможны проблемы с теми BDE-приложениями, которые используют другие механизмы управления базами данных, — могут возникнуть большие трудности при модернизации. В этом случае щелкните на кнопке Settings и перейдите к приведенным ниже этапам работы мастера установки BDE (BDE Setup Wizard), чтобы установить псевдоним (псевдонимы) BDE, которые понадобятся вашему приложению.

1. Щелкните на кнопке Next и добавьте новый псевдоним. Введите такое же имя псевдонима, как в вашем компьютере.
2. Установите путь, тип и необходимые параметры конфигурации для каждого включаемого вами псевдонима. Обычно файлы базы данных помещают в папку, которая вложена в папку инсталляции (и это правильно!). Обозначьте это вложение такой префиксной составляющей пути, как `<INSTALLDIR>`. Например, если бы я инсталлировал файлы базы данных в подпапке папки инсталляции, я бы ввел путь в таком виде: `<INSTALLDIR>\databases`.
3. Щелкните на кнопке Finish в последнем диалоговом окне мастера, и в системном реестре компьютера-приемника будут сконфигурированы нужные записи, а также будут подготовлены все нужные файлы.

Чтобы увидеть все включенные вами файлы, можете заглянуть во вкладку Advanced (Дополнительно).

Раздел Specify Components and Files (Указание компонентов и файлов)

Программа InstallShield собирает файлы в логические группы. Вы можете назвать эти группы по своему усмотрению, хотя при этом не мешает подумать о своем же удобстве. В течение ближайших пяти минут вы, вероятно, еще будете помнить, что означает такое имя группы, как `B1`, но не подведет ли вас память через несколько недель, когда вы захотите обновить свое приложение? Здесь позаботьтесь исключительно о себе, ведь группы файлов конечный пользователь не увидит.

Все файлы одной группы при инсталляции вашего приложения будут помещены пользователем в одну папку. Обычно для параметра File Update Method (Метод обновления файлов) в разделе Properties оставляют значение, предложенное по умолчанию. У вас должна быть группа файлов приложения. Могут быть дополнительные группы для компонентов, подготовленных ранее. Если вы будете использовать BDE, необходимо создать группу файлов для базы данных. Имя приемника должно совпадать с тем, что вы ввели во время работы с мастером установки BDE (BDE Setup Wizard)

Если у вас есть системные файлы Windows (например, DLL- и OCX-файлы), вы должны назначить для них в качестве приемника значение `<WINSYSDIR>`. Не стоит забывать о `.dll`-библиотеках, импортированных в ваш проект.

Чтобы распределить файлы по соответствующим группам, можно воспользоваться методом “перетащить и опустить” в окне программы Windows Explorer.

Для запуска программы Explorer, если вы еще не запустили ее, щелкните на кнопке Launch Explorer (Запуск Explorer) во вкладке Groups (Группы) диалогового окна Specify

Components and Files. Измените размер окна Explorer так, чтобы можно было перетаскивать файлы. Затем перетащите их в соответствующие группы.

Чтобы создать новую группу, щелкните на кнопке New Group (Создать группу), введите имя новой группы, а затем выберите новую папку инсталляции, если она не совпадает с предложенной по умолчанию.

Раздел Select User Interface Components (Выбор компонентов интерфейса пользователя)

Диалоговое окно Components (Компоненты) позволяет создавать и модифицировать компоненты (строительные блоки пользовательской настройки), а также подготовить их описание, чтобы помочь пользователю работать с ними.

Мы подошли к довольно забавной части программы InstallShield. В этом разделе нужно указать, в каком виде программу инсталляции увидит конечный пользователь. Здесь у вас большой простор для выбора различных деталей.

Неплохо бы убедиться, что включаемые вами диалоговые окна соответствуют приложению, которое вы собираетесь инсталлировать. Не следует полагаться на значения, устанавливаемые по умолчанию — они ведь попросту могут не подойти вам.

Растровое изображение приветственного экрана всегда выглядит профессионально, но важно найти компромисс в размерах. Добавление растрового изображения в программу инсталляции неизбежно увеличит ее размер, и если вы собираетесь распространять ее через Internet, этот фактор следует обязательно принять во внимание. Однако если ваше приложение будет распространяться посредством компакт-дисков, размер не имеет существенного значения.

У вас также должно быть место для документов, предназначенных для первоначального ознакомления (файлы Readme) и касающихся лицензирования программных продуктов. Многие производители бесплатных и условно-бесплатных Internet-приложений советуют помещать такую информацию в программу инсталляции, а также в основной Zip-файл вашего приложения. Поскольку печальная истина состоит в том, что многие пользователи не читают такого рода информацию, то, если вы предоставите им шанс сделать это во время ожидания, они не будут на вас в обиде.

Не забывайте, что текст, включаемый вами в опции программы инсталляции, необходимо отредактировать таким образом, чтобы ширина страницы не превышала 65 символов; в противном случае нет никакой гарантии, что ее форматирование будет соответствовать вашим намерениям. Я, например, использую старый добрый редактор Notepad, но не забываю отключить режим переноса не умещающихся слов на следующую строку.

У вас также есть возможность использовать диалоговые окна Setup Type (Тип установки) и Custom Setup (Пользовательская установка). Если вы не используете несколько компонентов установки приложения, эти диалоговые окна вам не нужны.

Если вдруг вы почувствуете, что какая-нибудь опция установки выглядит недостаточно хорошо, чтобы понравиться пользователю, щелкните на кнопке Preview и взгляните на оформление диалогового окна. Оно не будет в точности соответствовать созданному вами, но все же вы получите общее представление.

Почти для всех приложений имело бы смысл включить диалоговое окно Program Folder (Папка программы). Эта папка предназначена для хранения пиктограмм приложения. При желании пользователь сможет изменить предлагаемое имя этой папки.

Обычно в программу инсталляции включают диалоговые окна Start Copy (Начать копирование) и Progress (Отчет о ходе выполнения). Они вселяют в пользователя уверенность, что процесс инсталляции приложения не завис, а еще пока “дышит”.

Информационные экраны — прекрасный способ рассказать пользователям о характеристиках инсталлируемого программного продукта и других продуктах вашей компании. Ин-

формационные экраны представляют собой растровые изображения с одним и тем же именем, за которым следует некоторое число. Они будут появляться на экране в процессе инсталляции, но при этом следует иметь в виду следующее.

- Размеры приложения. Чем больше растровых изображений вы включите, тем больше станет размер программы инсталляции.
- Длительность процесса инсталляции. Если для инсталляции приложения требуется лишь несколько секунд, то ни о каких информационных экранах не может быть и речи, поскольку пользователь просто не успеет их прочитать.

Диалоговое окно Setup Complete (Установка приложения завершена) используется не столько для того, чтобы констатировать факт завершения инсталляции, сколько для того, чтобы напомнить о необходимости перезагрузки компьютера перед использованием приложения, если в него включены `.dll`-, `.ocx`- или любые другие пакеты времени выполнения, которые не были обновлены во время инсталляции.

Раздел Make Registry Changes (Внесение изменений в реестр)

В этом разделе можно записать в системный реестр информацию, которая потребуется приложению (например, действующие по умолчанию значения, которые запоминаются в реестре вашим приложением и используются при первом его запуске, или любая другая информация, которую нужно хранить в реестре).

Для этого достаточно выбрать ключевой раздел, используемый вами в приложении (со всеми необходимыми значениями). Ниже приводится метод добавления ключа в программу инсталляции. На рис. 29.5 показан результат добавления ключа в системный реестр.

```
HKEY_CURRENT_USER\CBUILDER\TEST\INSTALL
```

Щелкните на ключе `HKEY_CURRENT_USER`, который уже доступен для вас, а затем — на кнопке Add Key (Добавить ключ). После этого введите первую часть нового ключа `CBUILDER` и все остальные, пока не получите все реестровые адреса, необходимые для добавления значений.

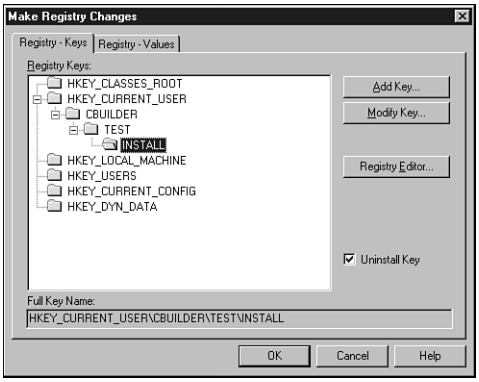


Рис. 29.5. Установка стандартных ключей и значений в системном реестре

Если вы допустили ошибку, можете для изменения текста использовать кнопку Modify Key (Модифицировать ключ) либо просто выберите ключ с ошибкой и удалите его.

Если вы хотите, чтобы добавленные вами в реестр записи были ликвидированы средствами удаления программы InstallShield, просто установите флажок опции Uninstall Key (Удалить ключ).

Чтобы добавить параметр реестра, сначала убедитесь, что вы выбрали именно тот ключ, к которому собираетесь добавить этот параметр, откройте вкладку Values и введите имя параметра вместе с его значением по умолчанию. Необходимо также выбрать тип используемого значения (`String`, `Binary`, `DWORD He`). В большинстве случаев используется тип `String`, но если у вас используется другой тип, вы будете об этом знать еще на этапе разработки приложения.

Раздел Specify Folders and Icons (Указание папок и пиктограмм)

Это диалоговое окно позволяет задать пиктограммы и папки, которые ваша программа будет использовать в меню Windows Start (Пуск) после инсталляции. Если в окне Visual Design вы выбрали выполняемый файл, он появится здесь автоматически. Любые пиктограммы, которые вы хотите добавить, в том числе и стандартные, должны быть добавлены вручную; в противном случае при инсталляции пользователем вашего приложения они на экране не появятся.

Для этого щелкните на кнопке, расположенной справа от поля Run Command (Команда "Выполнить"), и из предложенного списка выберите файл, для которого хотите создать пиктограмму. Введите, если нужно, любые параметры запуска, добавьте текст (в поле Description), который будет отображен под пиктограммой и щелкните на кнопке Add Icon (Добавить пиктограмму). Этот процесс необходимо повторить для всех пиктограмм вашего приложения. С большинством файлов (например, Readme и файлов справки) связаны стандартные пиктограммы.

Во вкладке Advanced (Дополнительно) у вас есть возможность выбрать место для размещения пиктограмм. Обычная их "среда обитания" — меню Start⇨Program (Пуск⇨Программы). Можно также разместить их в меню Start (Пуск). Опцию Desktop (Рабочий стол) следует использовать с осторожностью, но и такой вариант возможен, особенно в случае, если пользователь практически незнаком с компьютером или приложение будет постоянно в работе. Однако не забывайте, что слишком большое количество пиктограмм на рабочем столе создает неразбериху, в которой трудно найти нужный ярлык. Установка опции Startup (Запуск) обеспечит запуск вашей программы после перезагрузки компьютера или входе в систему пользователя. Опция Send To (Отправить) поместит ваше приложение в меню Send To, которое часто используется приложениями.

Раздел Run Disk Builder (Создание программы инсталляции)

Теперь мы подошли к сути InstallShield. Именно здесь все файлы вашего проекта собираются в единое целое и превращаются в программу инсталляции — подобно проекту в C++Builder.

Сначала выберите подходящий тип носителя данных. Если вы выберете тип 1.44MB, InstallShield разделит вашу программу инсталляции на папки размером 1,44 Мбайт, готовые к копированию на дискету (дискеты).

Если вы хотите сделать однофайловую инсталляцию, выберите в качестве типа носителя данных вариант CD-ROM (компакт-диск). Выбрав тип носителя, щелкните на кнопке Build (Построить), и программа инсталляции будет построена за вас. Если в процессе построения вы получите ошибки, исправьте их и снова щелкните на кнопке Build. Проект следует сохранить, чтобы впоследствии в него можно было внести изменения — добавить или удалить файлы, либо вообще изменить что-то. После этого снова вернитесь к этому разделу, выберите тип носителя и щелкните на кнопке Build — в этот момент можно считать сделанной львиную долю работы, ведь вы создали идеальную основу для будущих обновлений!

Раздел Create Distribution Media (Создание дистрибутива)

Если вы собираетесь распространять свое приложение посредством дискеты (дискет), можете использовать это диалоговое окно, чтобы сделать для себя настоящие дисковые копии. Предлагаемые здесь опции чрезвычайно просты. Выберите папку или устройство, на которое вы хотите скопировать дискеты, и щелкните на кнопке Copy All Disk Images (Скопировать все копии дисков).

Следует отметить, что эта часть программы InstallShield — необязательна для выполнения, поскольку дисковые копии уже были созданы и сохранены вами в папке, указанной при создании проекта инсталляции. Все, что вам осталось сделать, — скопировать на свою дискету содержимое каждой папки: помеченной именем `Disk1` — для диска 1, `Disk2` — для диска 2 и т.д. Только сначала не мешало бы убедиться в том, что у вас достаточно дискет, отформатированных с использованием соответствующих меток.

Версия приложения InstallShield Express, которая поставляется вместе с C++Builder 5, не включает варианта создания однофайлового выполняемого модуля. Если вас интересует именно этот вариант, купите пакет Package for the Web у корпорации InstallShield (`http://www.InstallShield.com`). Можно также использовать такой пакет, как WinZip или PK-Zip. Коммерческие версии обеих программ позволяют создавать самораспаковывающиеся выполняемые программы и запускать программу из архива после распаковки. Этот метод вам потребуется почти наверняка, если вы собираетесь распространять свое приложение через Internet, поскольку чем меньше общий размер приложения, тем больше вероятность, что его кто-то захочет загрузить на свой компьютер.

Тестирование

После создания программы инсталляции необходимо ее протестировать на каком-нибудь компьютере. Я всегда сначала тестирую ее на том же компьютере, где находится проект инсталляции, для того, чтобы убедиться в работоспособности пользовательского интерфейса (растровые изображения, формат файла Readme и пр.). Затем, если необходимо внести некоторые изменения, я делаю это без проблем, перестраиваю проект, снова тестирую и так до тех пор, пока, наконец, не буду полностью удовлетворен — не забывая, однако, удалять то, что было установлено вначале.

Удовлетворившись состоянием программы инсталляции, можете перейти к другому компьютеру и по-настоящему проверить работу программы инсталляции своего приложения. Идеальным вариантом было бы инсталлировать продукт на чистую версию Windows, но, к сожалению, мы живем не в идеальном мире.

Не переживайте, если инсталляция у вас не работает с первого раза; разберитесь, в чем дело, а затем вернитесь к своему компьютеру, на котором остались файлы проекта, внесите необходимые исправления, перестройте проект, снова опробуйте его и повторяйте этот цикл до тех пор, пока все не станет на свои места.

После успешной инсталляции приложения стоит затратить некоторое время на то, чтобы убедиться в нормальной работе приложения и корректной установке всех файлов (компонентов). В идеале следовало бы выполнить инсталляцию на различные диски, в различные папки и с различными вариантами установки компонентов (если таковые предусмотрены). На это может уйти много времени, но это время не будет потрачено зря. Ничего нет хуже разгневанного пользователя, который не может инсталлировать ваше приложение, загрузив его из недр Internet.

Теперь вы знаете, как пользоваться программой InstallShield Express. Жизнь не раз подтверждала, что любую программу (и эту в том числе) стоит опробовать еще до того, как она действительно вам понадобится, и причем “на вчера”. Стрессовое состояние цейтнота еще никому на пользу не шло: ни приложению, ни его разработчику.

Резюме

Цель этой главы (как и предыдущей) — дать представление об инсталляции приложений, маркетинге, распространении и поддержке программных продуктов. Несмотря на то что все эти методы рассматривались со ссылкой на C++Builder, они применимы к программным продуктам, разрабатываемым с помощью любых RAD-средств. Использование этих методов может значительно улучшить качество приложений и поднять их в глазах пользователей.

Мы рассмотрели методы инсталляции и поддержки приложений с помощью программы Install Maker (компании ClickTeam), CAB- и INF-файлов, а также посредством программы InstallShield Express. Кроме того, мы затронули вопросы поддержки программных продуктов, акцентируя внимание на создании и распространении обновлений и заплат. Наконец, мы рассмотрели новую систему управления версиями TeamSource, включенную в поставку C++Builder 5 Enterprise.

База знаний

ЧАСТЬ

VI

СОВЕТЫ, ПРИЕМЫ И РЕКОМЕНДАЦИИ

РЕАЛЬНЫЙ ПРИМЕР

Глава

30

Советы, приемы и рекомендации

Пит Педерсен
Халид Алманай
Пол Густавсон
Филипп Х. Блантон II
Джон Мак-Суин

Как упоминалось во введении к этой книге, мы начали с опроса разработчиков C++Builder, чтобы попытаться понять, что именно они хотели бы найти в этой книге. Сразу стало ясно, что многие хотели бы получить больше советов по примеру замечательной книги *Borland C++Builder How-to*. Поэтому основу настоящей главы составили советы и рекомендации, которые охватывают широкий диапазон полезных методов работы в среде C++Builder.

Разделы этой главы написаны различными авторами, поэтому и различия в стиле написания отдельных разделов здесь более ощутимы, чем в других главах. Одни разделы содержат полное описание, призванное помочь вам в создании законченного полнофункционального приложения, в других — лишь намечается структура программы и предлагаются пути ее усовершенствования, а в третьих акцент ставится на внутренней работе примеров программ (представленных на компакт-диске, прилагаемом к этой книге), а не на действиях, которые нужно выполнить для создания полнофункционального приложения.

Поскольку разделы этой главы охватывают множество различных и не связанных между собой тем, читателю необязательно работать с ними от начала главы и до конца. Но если вы предпочитаете именно последовательный стиль работы с книгой, не забывайте, что каждый раздел этой главы представляет отдельное приложение, а посему, приступая к новому разделу, всегда закрывайте свой текущий C++Builder-проект (для этого из главного меню C++Builder выберите сначала команду File⇨Save All (Файл⇨Сохранить все), а затем — команду File⇨Close All (Файл⇨Закрыть все)), после чего с помощью команды File⇨New Application (Файл⇨Создать приложение) можете начинать работу с новым приложением.

Как заставить клавишу <Enter> имитировать клавишу <Tab>

Вам, очевидно, встречались приложения, где пользователю предлагается указать, будет ли клавиша <Enter> действовать в качестве клавиши <Tab>. Такая "подмена" бывает весьма удобна для тех, кто вводит данные и закрепляет навыки использования клавиши <Enter>, расположенной на вспомогательной клавиатуре. Необходимость использовать клавишу <Tab> заставляет оператора напрягаться, что приводит к снижению производительности. В этом разделе мы рассмотрим, как реализовать эту функцию с помощью C++Builder.

Решение

Булево свойство `KeyPreview` класса `TForm` предоставляет возможность перехватывать все нажатия клавиш, посылаемые с клавиатуры, до того как форма получит право реагировать на них. При истинном значении свойства `KeyPreview` форма сгенерирует события `OnKeyPress` и `OnKeyDown` прежде, чем имеющий фокус (т.е. активный) компонент получит возможность генерировать собственные события, тем самым позволяя форме "предвидеть" каждое нажатие клавиши. Обработчики этих событий называются `FormKeyPress` и `FormKeyDown`. Для примера мы воспользуемся событием `FormKeyPress`.

В списке параметров обработчика события `FormKeyPress` нетрудно заметить ссылку на переменную типа `char`, именуемую `Key`. Эта переменная содержит значение каждого нажатия клавиши, производимого пользователем при вводе данных в форму. Суть метода состоит в анализе этой переменной при каждом нажатии клавиши и игнорировании всех ее значений, за исключением значения клавиши <Enter>. При обнаружении этой клавиши имитируется клавиша <Tab> и обнуляется значение переменной `Key` перед его дальнейшей передачей.

Комментарии к программе

Для создания приложения TabDemo выполните следующие действия.

1. Запустите C++Builder и создайте новое приложение.
2. Сохраните приложение, щелкнув на кнопке с изображением стопки дискет, расположенной на панели инструментов C++Builder. Модуль формы сохраните под именем `MainUnit.cpp`, а файл проекта — под именем `TabDemo.bpr` .
3. Добавьте в форму флажок (компонент `TCheckBox`), разместив его в верхней части формы.
4. Под этим флажком расположите вертикально шесть надписей (компонент `TLabel`).
5. Вставьте рядом с каждой надписью по строке редактирования (компонент `TEdit`), используя рис. 30.1 в качестве примера макета формы.
6. Наконец, поместите в нижнюю часть формы одну кнопку (компонент `TButton`).

Для того чтобы ваше приложение выполняло инициализацию с пустыми полями для ввода данных, вам стоит удалить значение `Text` каждой строки редактирования (`TEdit`).

Чтобы быстро поместить в форму сразу несколько компонентов, нажмите и удерживайте клавишу <Shift> в процессе их выбора из палитры компонентов. При этом вокруг выбранного компонента вы должны заметить появление маленькой синей рамки. Каждый щелчок на форме будет означать размещение в указанной позиции соответствующего компонента. По завершении добавления компонентов этот режим можно отменить, щелкнув на кнопке со стрелкой, расположенной также на палитре компонентов.

Установите свойства каждого компонента в соответствии с табл. 30.1.

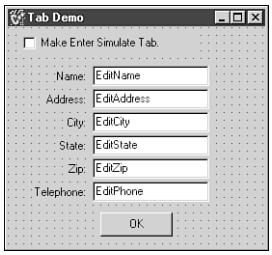


Рис. 30.1. Приложение TabDemo в режиме разработки

Таблица 30.1. Значения свойств главной формы приложения TabDemo

Компонент	Свойство	Значение
`Tform`	`Name`	`FormMain`
	`KeyPreview`	`true`
`TCheckBox`	`Name`	`CBSimulateTab`
	`TabStop`	`false`
	`Caption`	`Make Enter Simulate Tab`
	`Checked`	`false`
	`TabOrder`	`0`
`TLabel`	`Name`	`Label1`

Окончание табл. 30.1

Компонент	Свойство	Значение
	Caption	Name:
TLabel	Name	Label2
	Caption	Address:
TLabel	Name	Label3
	Caption	City:
TLabel	Name	Label4
	Caption	State:
TLabel	Name	Label5
	Caption	Zip:
TLabel	Name	Label6
	Caption	Telephone:
TEdit	Name	EditName
	TabOrder	1
TEdit	Name	EditAddress
	TabOrder	2
TEdit	Name	EditCity
	TabOrder	3
TEdit	Name	EditState
	TabOrder	4
TEdit	Name	EditZip
	TabOrder	5
TEdit	Name	EditPhone
	TabOrder	6
TButton	Name	BtnOK
	Caption	OK
	TabStop	false

Не забудьте проверить ширину значения `CBSimulateTab`, чтобы убедиться в том, что заголовок виден полностью. В отличие от надписей, ширина флажков не настраивается автоматически по размеру их заголовков.

Если вам не нравится внешний вид этого демонстрационного приложения, можете менять взаимное расположение и размеры компонентов в форме до тех пор, пока не начнете получать от нее эстетическое удовольствие. Однако имейте в виду, что при изменении порядка расположения компонентов необходимо также изменить порядок табуляции, в противном случае поведение формы будет несколько “непоследовательным”. При нажатии клавиши <Tab> Windows передает фокус следующему компоненту в соответствии со значением свойства `TabOrder`, присущего каждому компоненту, поэтому после реорганизации формы следует убедиться в правильности расстановки этих значений.

Чтобы быстро и просто изменить порядок табуляции, щелкните правой кнопкой мыши на контейнере компонента (в данном случае — на главной форме) и выберите из контекстного меню команду Tab Order (Порядок табуляции).

После "изготовления" формы переходим к программированию обработчиков событий. Создайте обработчик событий для кнопки OK (BtnOK), дважды щелкнув на этой кнопке. Введите в обработчик событий следующую строку программы (она говорит сама за себя).

```
Close();
```

Теперь в окне Object Inspector выберите главную форму (FormMain), либо щелкнув на пустом месте формы, либо выбрав ее из комбинированного списка, расположенного в верхней части окна инспектора объектов. Щелкните на вкладке Events (События), а затем дважды щелкните на событии OnKeyPress, чтобы создать его обработчик. Введите в этот обработчик следующую программу.

```
if (Key == VK_RETURN && CBSimulateTab->Checked)
{
    Key = 0;
    SelectNext(FormMain->ActiveControl, true, true);
}
```

Если форма FormMain активна (получила фокус), то этот код выполняется при каждом нажатии пользователем любой клавиши. Сначала проверяем, не равно ли значение, содержащееся в переменной Key, константе VK_RETURN (означающей, что была нажата клавиша <Enter>), определенной в заголовочном файле winuser.h. Если это равенство подтвердилось, оцениваем булево свойство Checked флажка с именем CBSimulateTab, чтобы узнать, подтвердил ли пользователь свое намерение реализовать имитацию клавиши <Tab> клавишей <Enter>. Если два эти условия оказались равными значению true, выполняем следующие действия.

- Устанавливаем значение переменной Key равным 0, которое предотвратит реакцию приложения на нажатие клавиши <Enter>.
- Вызываем VCL-метод SelectNext(), который передаст фокус следующему элементу управления, согласно значению свойства TabOrder формы.

Метод SelectNext() принимает в качестве параметров текущий (активный) элемент управления и два булевых значения. Первое булево значение (true) означает ваше желание перейти к следующему элементу в tab-последовательности. Передача значения false в данном случае означала бы смещение фокуса в обратном направлении (или вверх) по списку элементов управления формы. Второе булево значение определяет ваше намерение учесть свойство TabStop следующего элемента управления при изменении фокуса. Если здесь передать значение false, этот метод проигнорирует свойство TabStop и "в приказном порядке" передаст фокус следующему элементу по списку TabOrder. Подробнее о методе SelectNext можно прочитать в справочном руководстве по VCL (VCL Reference Guide).

Сохраните, скомпилируйте и выполните новое приложение. Если компилятор не обнаружит ошибок, вы сможете переходить от одного поля формы к другому (в порядке, заданном tab-последовательностью) с помощью клавиши <Enter> до тех пор, пока установлен флажок CBSimulateTab.

Константа VK_RETURN соответствует лишь одному из множества виртуальных кодов клавиш Windows. Она определена в заголовочном файле winuser.h, как показано ниже.

```
#define VK_RETURN 0x0D
```

Если вы разбираетесь в шестнадцатеричной системе счисления, то сразу поймете, что 0x0D — это шестнадцатеричный эквивалент десятичного числа 13. Вы могли бы сказать, что, мол, можно было сразу сравнить значение переменной Key с числом 13 (if (Key == #13)), но такой стиль программирования не считается профессиональным. Возьмите в привычку определять все константы своей программы в одном месте, а затем используйте их соответствующим образом. Уже одно это повысит уровень ваших программ, не говоря уже о том, что их поддержка станет намного проще. В свободное время вам стоит взглянуть и на другие виртуальные коды клавиш Windows в заголовочном файле winuser.h или обратиться за дополнительными сведениями об этих кодах к справочнику программиста по Win32API. Этот пример — лишь "намек" на большие потенциальные возможности анализа и обработки нажатий клавиш.

Немного о „ловушках"

Иногда работа некоторых программ не оправдывает наших ожиданий. Поэтому стоит заранее узнать о причинах подобных явлений.

Почему нельзя просто присвоить значение константы VK_TAB переменной Key

Когда система Windows получает от клавиатуры Tab-команду (в результате нажатия клавиши <Tab>), она немедленно отправляет активному окну сообщение WM_NEXTDLGCTL, "считает" клавишу обработанной и не передает информацию об этом нажатии клавиши вашему приложению. Следовательно, вам нужно самим в явном виде переместить фокус, а не просто установить значение переменной Key равным константе VK_TAB. А впрочем, убедитесь сами. Ваше приложение проигнорирует это присвоение.

Установив для свойства Default кнопки BtnOK значение true, я не могу работать!

Так и должно быть. Если активная форма содержит кнопку, действующую по умолчанию, Windows "в обход" формы будет передавать нажатие клавиши прямо этой кнопке, считая для себя обработку нажатия клавиши законченной. В этом случае форма никогда не получит возможность обработать его "по-своему". Поэтому не следует смешивать кнопки, действующие по умолчанию, и "переориентацию" клавиши <Enter>.

TMemo-поля почему-то не работают у меня должным образом

Обычно приходится прилагать чуть больше усилий, если форма содержит компоненты, в работе которых предусмотрено нажатие пользователем клавиши <Enter> (это касается таких компонентов, как TMemo или TRichEdit). В таком случае вам нужно определить тип компонента, который имеет фокус в данный момент, и действовать соответственно. Следующий фрагмент программы представляет собой модифицированную версию обработчика события OnKeyPress, который будет корректно обрабатывать компоненты TMemo. Используя этот код,

вы сможете tab-способом войти в TMemo-поле, а уже внутри него клавиша <Enter> будет действовать как обычное сочетание кодов “возврат каретки”/“перевод строки” (CR/LF). Для выхода из TMemo-поля пользователям вашего приложения придется либо использовать настоящую клавишу <Tab>, либо щелкнуть на следующем компоненте. Опустите компонент TMemo в форму, определите tab-последовательность для компонентов формы и опробуйте предложенный вариант кода.

```
if (Key == VK_RETURN && CBSimulateTab->Checked)
{
    if (String(
      FormMain->ActiveControl->ClassName()).AnsiCompare("TMemo"))
     {
          Key = 0;
          SelectNext(FormMain->ActiveControl, true, true);
     }
}
```

Резюме по имитации клавиши <Tab>

Заставить клавишу <Enter> имитировать клавишу <Tab> — относительно просто. Анализ виртуальных кодов клавиш Windows по мере их поступления от клавиатуры позволяет разработчику создать механизм реагирования на множество условий и тем самым создать очень “чуткое” и интеллектуальное приложение, которое на один шаг будет “опережать” пользователя. В реальных приложениях часто используется какой-нибудь элемент меню или отдельное окно настройки, предлагающее пользователю сделать выбор между стандартно работающей клавишей <Enter> и имитатором tab-переходов.

Определение версии операционной системы

По мере эволюции операционной системы Microsoft Windows создавались ее различные версии, которые отличались одна от другой некоторыми особенностями поведения. Например, Windows 95 и 98 по умолчанию размещаются в папке Windows, в то время как Windows NT версии 3.51 и 4.0, а также Windows 2000 размещаются по умолчанию в папке Winnt. Поскольку эти различия влияют на поведение пользовательских приложений, возникает необходимость в определении версии Windows, в среде которой работает приложение.

Решение

Win32 API-функция GetVersionEx() предоставляет разработчику возможность легко получать информацию об операционной системе, в среде которой будет работать его приложение. С помощью этой API-функции легко определить версию операционной системы (OS). На рис. 30.2 и рис. 30.3 показан результат работы приложения OSVer, запущенного под управлением двух различных операционных систем.

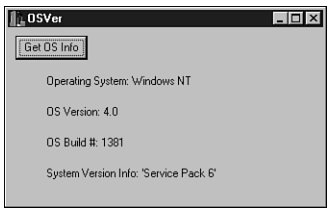


Рис. 30.2. Приложение OSVer, работающее в среде NT 4.0

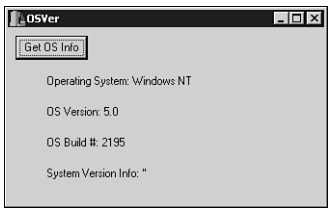


Рис. 30.3. Приложение OSVer, работающее в среде Windows 2000 (NT 5)

Описание приложения

Для создания приложения OSVer выполните следующие действия.

1. Запустите C++Builder и создайте новое приложение.
2. Сохраните приложение, щелкнув на кнопке с изображением стопки дискет, расположенной на панели инструментов C++Builder. Модуль формы сохраните под именем `MainUnit.cpp`, а файл проекта — под именем `OSVer.bpr` .
3. Поместите в форму кнопку (компонент `TButton`), расположив ее где-нибудь в верхней части формы, назовите ее `BtnGetInfo` и установите ее заголовок равным значению `Get OS Info`.
4. Под этой кнопкой разместите вертикально четыре надписи (компоненты `TLabel`).

Подготовив форму, переходим к программированию обработчиков событий. Создайте обработчик событий для кнопки `BtnGetInfo`, дважды щелкнув на этой кнопке. Введите в ее обработчик содержимое листинга 30.1.

Листинг 30.1. Обработчик щелчков на кнопке для приложения OSVer

```
void __fastcall TForm1::BtnGetInfoClick(TObject *Sender)
{
    OSVERSIONINFO osinfo;
    osinfo.dwOSVersionInfoSize = sizeof(osinfo);
    GetVersionEx(&osinfo);

    if (osinfo.dwPlatformId == VER_PLATFORM_WIN32s)
        Label1->Caption = "Operating System:Windows Win32s";
     else if (osinfo.dwPlatformId == VER_PLATFORM_WIN32_WINDOWS)
          Label1->Caption = "Operating System:Windows 9x";
     else if (osinfo.dwPlatformId == VER_PLATFORM_WIN32_NT)
          Label1->Caption = "Operating System:Windows NT";
     else Label1->Caption = "Operating System:Unknown";

     Label2->Caption = String("OS Version: ")
                              + String(osinfo.dwMajorVersion)
                              + "." + String(osinfo.dwMinorVersion);

     Label3->Caption = String("OS Build #: ")
                                     + String(osinfo.dwBuildNumber);
```

```
    Label4->Caption = String("System Version Info:‘")
                                        + osinfo.szCSDVersion + "’";
}
```

Скомпилируйте и выполните это приложение. После щелчка на кнопке Get OS Info (Получить информацию об операционной системе) на форме появятся данные о версии операционной системы.

В заголовочном файле `winbase.h` определены константы `VER_PLATFORM_WIN32s`, `VER_PLATFORM_WINDOWS` и `VER_PLATFORM_WIN_NT`. Стоит взглянуть на их определение и с помощью средств поиска в исходных файлах вашей системы поинтересоваться, как другие используют эту информацию.

Например, полезно рассмотреть код конструктора `TMouse` (файл `SysUtils.pas`). Именно здесь разработчики компании Borland используют информацию об операционной системе для определения того, нужно ли активизировать встроенную поддержку мыши со скроллингом (Wheel-Mouse). Однако имейте в виду, что библиотека VCL Borland написана на языке Object Pascal, который несколько отличается от C или C++. Основное отличие в данном фрагменте программы состоит в том, что оператор = является оператором сравнения, а не присваивания.

```
// Мышь со скроллингом внутренне поддерживается в Windows 98 (и более поздних
// версиях), а также в Windows NT 4.0 (и более поздних версиях).
FNativeWheelSupport :=
    ((Win32Platform = VER_PLATFORM_WIN32_NT) and (Win32MajorVersion >= 4))
    or ((Win32Platform = VER_PLATFORM_WIN32_WINDOWS)
    and ((Win32MajorVersion > 4) or ((Win32MajorVersion = 4)
    and (Win32MinorVersion >= 10))));
```

Переменная `FNativeWheelSupport` устанавливается равной значению `true`, если в качестве операционной системы используется Windows NT 4.0 или более поздняя версия, что обеспечивается выполнением следующей строки программы.

```
((Win32Platform = VER_PLATFORM_WIN32_NT) and (Win32MajorVersion >= 4)
```

Переменная `FNativeWheelSupport` также устанавливается равной значению `true`, если в качестве операционной системы используется Windows 98 (Windows версия 4.10) или более поздняя версия, что обеспечивается выполнением следующих строк программы.

```
(Win32Platform = VER_PLATFORM_WIN32_WINDOWS) and ((Win32MajorVersion > 4)
  or ((Win32MajorVersion = 4) and (Win32MinorVersion >= 10)))
```

Резюме по определению версии операционной системы

Получение информации об операционной системе — относительно простая задача. Поскольку в работе разных версий операционных систем Windows существуют различия, то такая информация необходима для обеспечения полной поддержки вашим приложением всех платформ Win32. Способность приложения определить, в какой операционной системе оно выполняется, позволит корректно использовать ресурсы каждой системы.

Программирование с использованием чисел с плавающей запятой

Большинство программистов, занимаясь представлением десятичных чисел в двоичном коде, осведомлено о существовании определенных ограничений в использовании типов данных с плавающей запятой. Однако для многих, честно говоря, эта тема все же не вполне ясна. Цель этого раздела — кратко изложить идею, положенную в основу метода представления чисел с плавающей запятой. В следующих разделах будут намечены основные правила, соблюдение которых может существенно повлиять на точность вычислений по формулам, в которых используется тип данных с плавающей запятой.

Основа метода

Благодаря переносимости языка ANSI C++, числовой диапазон типов данных обычно ограничивается аппаратными возможностями используемого компьютера. Поскольку C++Builder предусматривает совместимость с IBM PC, схемы процессоров Intel 8088 и 8086 определяют способ, который применяется в C++Builder при использовании арифметики чисел с плавающей запятой. При проектировании процессора 8086 компания Intel пригласила к сотрудничеству трех специалистов (Кона (Kahn), Кунена (Coonan) и Стоуна (Stone)) для выработки формата чисел с плавающей запятой, который они смогли реализовать в 8087 FPU (Floating-Point Unit). Этот формат стали обозначать как *KCS Floating-point Format* (*K*ahn, *C*oonan, *S*tone), а позже он был принят (почти без изменений) в качестве стандарта IEEE представления чисел с плавающей запятой. Для удовлетворения требованиям по производительности и точности представления чисел компания Intel приняла три формата с плавающей запятой, встроенных в язык C++ (табл. 30.2).

Таблица 30.2. Числовой диапазон типов данных в языке C++

Имя	Тип данных в C++	Размер в битах	Числовой диапазон
С одинарной точностью (Single Precision)	`float`	32	8,43E–37–3,37E38
С двойной точностью (Double Precision)	`double`	64	1,7E–307–1,7E308
С расширенной точностью (Extended Precision)	`long double`	80	3,4E–4932–1,1E4932

В столбце *Числовой диапазон* указан диапазон для положительных чисел (для отрицательных подразумевается аналогичный диапазон). Все значения с плавающей запятой представляются посредством некоторого числа битов мантиссы и числа битов показателя степени (экспоненты). Для упомянутых выше типов данных разделение на эти категории битов показано в табл. 30.3.

Таблица 30.3. "Битовые" характеристики чисел с плавающей запятой

Тип данных	Общее число битов	Число битов мантиссы	Число битов экспоненты
`float`	32	24	8
`double`	54	53	11
`long double`	80	64	16

В файле float.h, который находится в подпапке Include главной папки C++Builder 5, можно увидеть все упомянутые выше типы, а также константы, содержащие значения, которые описывают компоновку каждого типа.

```
// Число битов, определяющих мантиссу в типе double.
#define DBL_MANT_DIG 53
// Число битов, определяющих мантиссу в типе float.
#define FLT_MANT_DIG 24
// Число битов, определяющих мантиссу в типе long double.
#define LDBL_MANT_DIG 64
```

Полное объяснение арифметических операций с плавающей запятой и способа представления в двоичном коде выходит за рамки этого раздела. Однако для понимания основных принципов все же будут изложены некоторые важнейшие концепции, затронутые в ходе описания.

Работа с числами

Чаще всего данными в формате с плавающей запятой оперируют в следующих случаях:

- при сложении и вычитании;
- при вычислении последовательностей операций сложения, вычитания, умножения и деления, заданных в формулах с использованием скобок;
- при сравнении данных.

Использование чисел с плавающей запятой ограничивается только представлением разработчика и требованиями клиента. Однако приведенный выше список охватывает достаточно широкий диапазон сфер приложений, поэтому можно вполне ограничиться этим списком, а рассмотрение каждого пункта с указанием на потенциальные опасности и полезными советами позволит вам достичь высокой точности при работе с такими форматами чисел.

Для начала рассмотрим кратко концепцию ошибок округления. Может показаться, что существует бесконечный диапазон точности, требуемой при работе с вещественными числами. Однако каждый разработчик должен четко осознавать, что практически всегда, когда он использует арифметику чисел с плавающей запятой в среде C++Builder (или в любой другой среде разработки), в вычисления вносятся ошибки. Это неизбежно.

Существуют две принципиальные причины, по которым число с плавающей запятой нельзя представить в двоичной форме: отсутствие приемлемого диапазона или достаточного количества битов, в которых будет содержаться число, а также то, что некоторые числа невозможно идеально преобразовать из одного формата в другой. Возьмем, например, дробь, равную одной трети. В десятичной форме она представляется в виде бесконечной дроби 0,3333… (с периодом повторения 3). Возьмем также одну десятую, которую в десятичной форме можно точно представить как 0,1, но в двоичной образуется период повторения вида 1,10011001100110011001101 × 2^–4. Если взять двоичные множители, образующие ряд дробей 1/2, 1/4, 1/8, 1/16, 1/32 и т.д., то можно заметить, что число 0,1 находится между 1/8 и 1/16. Следовательно, компьютер, чтобы как можно точнее представить 1/10, использует повторяющуюся и монотонно убывающую последовательность двоичных дробей.

```
Относительная погрешность = ε
ε = (реальное значение - вычисленное значение)/реальное значение
```

Выполнение сложения и вычитания

Можно показать, что даже при простом вычитании, когда оба числа используют полный диапазон точности, доступный в этом типе данных, результирующая ошибка может составлять 100% или ε = 1,0. Можно также показать, что, введя только один разряд защиты (вспомогательный младший разряд), можно значительно повысить точность. К счастью, стандарт IEEE гарантирует, что такая защита реализована в архитектуре компьютера, и вам не нужно беспокоиться об этом (см. файл `float.h`: `#define FLT_GUARD 1`). Вам показали, насколько существенными могут быть ошибки и что можно сделать в направлении их минимизации. В следующем примере число 9,95 вычитается из 10,2. Нормализуя эти числа с точностью до 2 десятичных разрядов, получаем следующее.

```
1,02×10^1-0,99×10^1=0,03×10^1=0,3 (см. ниже: верный ответ = 0,25)
```

Однако при добавлении одного разряда защиты получаем следующее.

```
1,020×10^1-0,995×10^1=0,025×10^1 (правильный ответ).
```

Другой важный момент — нормализация. Числа с плавающей запятой нормализуются, что по сути означает сдвиг мантиссы таким образом, чтобы первая цифра (исключая знаковый разряд) содержала ненулевое значение. Возьмем, например, четырехзначное число 0,001E0, которое после нормализации принимает вид 1,000E–3. А теперь, если на самом деле у нас было число 1,001E–3, то представление в виде 0,001E0 привело бы к потере точности. Поэтому нормализация просто гарантирует, что при данном количестве доступных разрядов поддерживается максимальная точность. (Хотя это также означает, что нуль не может быть представлен в формате с плавающей запятой — об этом стоит подумать особо!)

При выполнении сложения и вычитания чисел с плавающей запятой (*f*loating-*p*oint *n*umbers, или FPN), необходимо сначала изменить их показатели степени, чтобы они стали равными. Конечно, компьютер делает это автоматически, но нетрудно убедиться в том, что посредством нормализации можно потерять любой выигрыш в точности, если существует значительная разница между показателями степеней двух чисел. Короче говоря, меньшее число подавляется.

Для демонстрации вышесказанного создайте новый проект (с помощью команды File⇨New Application) с тремя строками редактирования (компонент `TEdit`), тремя надписями (компонент `TLabel`) и одной кнопкой (компонент `TButton`), оставив всем им имена по умолчанию и введя в обработчик события `OnClick` кнопки `Button1` содержимое листинга 30.2. Разместите надписи `Label1`, `Label2` и `Label3` рядом со строками редактирования `Edit1`, `Edit2` и `Edit3`, соответственно. Установите свойство `Caption` надписи `Label1` равным `Increment`, а свойство `Caption` надписей `Label2` и `Label3` сделайте пустым (т.е. удалите заголовок, действующий по умолчанию). Наконец, установите свойство `Caption` кнопки `Button1` равным `Test Sum By Grouping` (Тестовое суммирование по группам) и убедитесь в том, что свойство `Text` строки редактирования `Edit1` установлено равным значению `0,00`, чтобы предотвратить любые начальные ошибки `EConvertError`.

Листинг 30.2. Программирование кнопки для сложения чисел с плавающей запятой

```
double sum1;
double sum2;
double increment;
int loops;
increment=Edit1->Text.ToDouble();
sum1=1.0E20;
sum2=0.00;
```

```
loops=100000000;
for (int i=0; i<loops; i++)
{
    sum1=sum1+increment;
    sum2=sum2+increment;
}
Label2->Caption="Simple Sum";
Label3->Caption="Improved Sum";
Edit2->Text=sum1;
Edit3->Text=1.0E+20+sum2;
```

Если теперь выполнить команду File⇨Save All (используя стандартные имена файлов), запустить эту программу и щелкнуть на кнопке Button1, то можно увидеть, что суммы, вычисляемые двумя методами, дают одинаковые результаты. Однако если последовательно увеличивать значение инкремента (как показано ниже) и перезапускать программу, вы убедитесь в постепенном росте ошибок. В табл. 30.4 и 30.5 показаны значения этих относительных погрешностей.

Таблица 30.4. Относительная погрешность при вычислении общей суммы для различных значений инкремента

Запуск	Инкремент	Значение sum1	ипсилон
1	0,001	1,00000000000000E+20	0,999999999999999
2	0,010	1,00000000000000E+20	0,999999999999990
3	0,100	1,00000000000000E+20	0,999999999999900
4	1,000	1,00000000000000E+20	0,999999999999000
5	1000	1,00000000000000E+20	0,999999999000000
6	10000	1,00000001638400E+20	1,000000006384000
7	1,E+08	1,00010000793600E+20	1,000000007935210
8	1,E+12	1,99999999590400E+20	0,999999997952000
9	1,E+14	1,01000000179677E+22	1,000000001778980

Таблица 30.5. Относительная погрешность при вычислении общей суммы для различных значений инкремента

Запуск	Инкремент	Значение sum2	ипсилон
1	0,001	1,00000000000000E+20	0,999999999999999
2	0,010	1,00000000000001E+20	1,000000000000000
3	0,100	1,00000000000010E+20	1,000000000000000
4	1,000	1,00000000000100E+20	1,000000000000000
5	1000	1,00000000100000E+20	1,000000000000000
6	10000	1,00000001000000E+20	1,000000000000000
7	1,E+08	1,00010000000000E+20	1,000000000000000
8	1,E+12	1,99999999741516E+20	0,999999998707580
9	1,E+14	1,01000000175909E+22	1,000000001741670

При просмотре текста этой программы нетрудно заметить, что для вычисления sum1 значение инкремента на каждой итерации цикла прибавляется к общему значению суммы, в то время как при вычислении sum2 накопление малых значений инкремента происходит во временной переменной, а затем (в конце) их сумма прибавляется к значению 1E20. Как видно из таблиц 30.4 и 30.5, относительные погрешности невелики. Однако второй вариант выглядит все-таки лучше. Изменение значений погрешности можно соотнести с процессом нормализации, при котором возможны потери сдвинутых цифр. Такой анализ приводит нас к первому эмпирическому правилу: порядок сложения чисел может влиять на результат. Поэтому всегда старайтесь складывать числа, показатели степени которых (в группах сложения) подобны. А затем для получения окончательного ответа сложите значения сумм групп.

Может показаться, что сделанный нами вывод основан исключительно на интуиции. Тогда рассмотрим проблему суммирования большого количества данных, значения которых очень сильно отличаются одно от другого по размерности, причем их группирование может осуществляться разными способами. Практика показывает, что многие не считают нужным сначала упорядочить числа, применив упомянутое выше правило для группирования чисел в соответствии с их размерностью, а затем суммировать значения сумм, полученных в группах. Однако иногда для поддержки приемлемого уровня точности без этого просто не обойтись. Часто предлагается другой метод — упорядочить числа, а затем суммировать их от меньшего к большему. Это минимизирует ошибку денормализации за счет минимизации относительной разности между текущим значением суммы и следующим прибавляемым к ней числом.

Однако существует более точный и более эффективный метод, именуемый формулой суммирования Кона (Kahn Summation Formula). Этот метод исправляет ошибку, связанную с потерей битов младшего порядка в значении инкремента.

```
//Теория суммирования Кона (Kahn Summation Theory).
S = X[1];
C = 0;
for (int i=2; i<N; i++)
{
    Y = X[i] - C;
    T = S + Y;
    C = (T -S) - Y;
    S = T;
)
```

Метод “Теория суммирования Кона” (Kahn Summation Theory) взят из статьи “What Every Computer Scientist Should Know About Floating Point Numbers”, написанной Дэвидом Голдбергом (David Goldberg) и опубликованной в журнале *Computing Surveys* в марте 1991 года.

Для его демонстрации возьмем приведенную выше программу вычисления значений sum1 и sum2, добавим в форму еще одну кнопку (компонент TButton), установив ее свойство Caption равным Kahn Summation. Введем в обработчик события Button2Click модифицированный код суммирования, содержащийся в листинге 30.3, затем выполним новый вариант программы и сравним все три идеи реализации суммирования.

Листинг 30.3. Метод суммирования, модифицированный на основе теории суммирования Кона, для предотвращения потери младших битов в данных

```
double sum3;
double sum4;
double increment;
double correction;
double temp_increment;
```

```
double temp_sum;
int loops;
sum3=0.00;
sum4=0.00;
increment=Edit1->Text.ToDouble();
loops=1E8;

correction=0.0;

for (int i=0; i<loops; i++)
{
    temp_increment = increment-correction;
    temp_sum = sum3 + temp_increment;
    correction = (temp_sum - sum3)- temp_increment;
    sum3 = temp_sum;
    sum4 = sum4 + increment;
}
Edit2->Text=sum3;
Edit3->Text=sum4;
Label2->Caption="Kahn Summation";
Label3->Caption="Simple Summation"
```

Вычисление значения `sum3` дало результат 10000 (метод Кона), а значения `sum4` — 9986,99998403165 (простое сложение). И хотя такой точности можно было бы легко добиться простым умножением инкремента на количество итераций в цикле, описанный метод можно применить к любому количеству произвольных неравных значений данных. Чтобы получить еще большее впечатление от идеи, позволяющей достичь точности вычислений, попробуйте выполнить это приложение, используя значение инкремента, равное числу 0,33333333. (Сумма Кона = 33333333, а стандартная сумма = 33333319,928899.)

Последовательность арифметических операций со скобками

Самым очевидным препятствием для получения результата при умножении и делении является переполнение (или потеря значимости). Такого рода опасности подстерегают обычно в следующих случаях.

- (Очень большое число) × (очень большое число) ведет к переполнению.
- (Очень малое число) × (очень малое число) ведет к потере значимости.
- (Очень большое число) / (очень малое число) ведет к переполнению.
- (Очень малое число) / (очень большое число) ведет к потере значимости.

Следовательно, при умножении чисел старайтесь упорядочивать их так, чтобы перемножались большие и малые числа. Однако при делении старайтесь делить числа подобного порядка.

"На руку" умножению и делению играет тот факт, что числа не нуждаются в денормализации — перед выполнением операции их не нужно преобразовывать к одинаковым показателям степени, поскольку в этом случае требуется лишь сложить и разделить соответствующие показатели степени. Таким образом, умножение и деление не страдают в такой степени от ошибки накопления, как сложение и вычитание. Отсюда следует, что при выполнении та-

кой последовательности вычислений со скобками, как (X+Y)(X–Y), вы получите более точные результаты в том случае, если раскроете скобки и сначала выполните умножение/деление, а затем сложение/вычитание (x^2 – y^2).

Сравнение данных

Наконец, мы часто сталкиваемся с проблемами при выполнении сравнений чисел с плавающей запятой (FPN). Как было показано выше, в определении конечного результата существенную роль играет метод вычисления, даже при идентичности FPN-компонентов. Следовательно, опасность непосредственного сравнения FPN-чисел на равенство и неравенство должна быть очевидной.

Чаще всего в области контроля границ говорят, что два числа равны, если разница между ними меньше или равна некоторому значению дельта (delta), определенному в виде конкретного диапазона (x равно y, если abs(x–y) <= delta).

Подобный псевдокод можно сгенерировать и для таких операторов, как <=, < и т.д.

```
// Простая функция для проверки равенства чисел с плавающей запятой
bool FloatTest (double Datum, double Test, double Error)
{
    double Temp;
    Temp = Datum - Test;
    if (fabs(Temp)<=Error)
        return true; //Возвращает true, если попадает в диапазон.
    else
        return false; // Возвращает false, если не попадает в диапазон.
)
```

В заключение о числах с плавающей запятой

Выполнение операций над числами с плавающей запятой является одной из тем, которые объединяются под общим заголовком *Числовые вычисления* (Numerical Computation). В вашем распоряжение множество легкодоступных источников; например, упомянутое выше издание Дэвида Голдберга вполне может служить в качестве отправной точки на этом пути. Надеемся, что эти общие замечания позволили вам представить, с какими проблемами вы можете столкнуться, используя числа с плавающей запятой, и указали некоторые пути повышения точности результатов вычислений.

Реализация экранных заставок

Экранные заставки становятся все более популярными, поскольку приложения имеют тенденцию к увеличению в размерах, и поэтому период инициализации (время загрузки форм в память) увеличивается. Они уведомляют пользователя о том, что в результате сделанного им двойного щелчка на пиктограмме приложения что-то происходит, убеждая его в ненужности повторять щелчок (наверное, подобный ход мыслей вам знаком: “Если оно не работает с первого раза, возможно, запуск стоит повторить!”).

Все приложения C++Builder состоят из функции `WinMain()`, которую можно найти в файле `<ProjectName>.cpp`. Эта функция является для приложений входной точкой для выполняемой программы (`<ProjectName>.exe`) и связана с установкой форм и присвоением формам глобальных переменных. Поскольку это единственная внешняя функция, отмечающая вход-

ную точку в программе, имеет смысл использовать ее параметры для передачи аргументов командной строки (или соответствующих имен файлов в окнах).

Следовательно, если мы собираемся создать экранную заставку, которую будет видеть пользователь до выполнения последующих действий, она должна находиться внутри этой функции. Поэтому, вместо того чтобы слепо вводить программные строки (их может быть приблизительно три), давайте лучше рассмотрим, что происходит при запуске приложения.

Функция WinMain()

После запуска C++Builder и добавления второй формы в стандартное приложение у нас имеется программный код, показанный в листинге 30.4 (см. файл Project1.cpp).

Листинг 30.4. Файл исходного кода проекта

```
#include <vcl.h>
#pragma hdrstop
USERES("Project1.res");
USEFORM("Unit1.cpp", Form1);
USEFORM("Unit2.cpp", Form2);
//---------------------------------------------------------------
WINAPI WinMain(HINSTANCE, HINSTANCE, LPSTR, int)
{
    try
    {
        Application->Initialize();
        Application->CreateForm(__classid(TForm1), &Form1);
        Application->CreateForm(__classid(TForm2), &Form2);
        Application->Run();
    }
    catch (Exception &exception)
    {
        Application->ShowException(&exception);
    }
    return 0;
}
```

С помощью параметров функции WinMain() передается информация о дескрипторе текущего и предыдущего экземпляра настоящей версии приложения и строковых значениях командной строки. Затем в теле try-блока для обработки исключительных ситуаций приложение начинает загрузку форм в память, как описано посредством файлов Form1.cpp и Form2.cpp.

Выражение Application->Initialize() инициализирует VCL-приложение для этого процесса.

Оператор Application->CreateForm() создает форму, заданную в параметрах функции CreateForm. С помощью первого параметра задается класс формы, а с помощью второго — объект формы. Операторы Application->CreateForm() C++Builder вводит для каждой формы, добавленной вами в проект. Эти операторы перечисляются в порядке добавления форм в проект, и в этом же порядке эти формы создаются в памяти во время выполнения приложения. Например, если скомпилировать и запустить это приложение, нетрудно заметить, что форма Form1 отображается в качестве главной формы. Однако, можно изменить содержимое файла Project1.cpp таким образом, что уже знакомые вам строки будут следовать в другом порядке.

```
Application->CreateForm(__classid(TForm2), &Form2);
Application->CreateForm(__classid(TForm1), &Form1);
```

После компиляции и выполнения измененного файла вы увидите Form2 в качестве главной формы. Ту же работу C++Builder автоматически сделает за вас, если главной формой назначить Form2 с помощью команды Project⇨Options (Проект⇨Параметры), в результате которой откроется диалоговое окно Forms (Формы).

Оператор Application->Run() заставляет процесс войти в цикл приема сообщений программы. С этого момента процесс начинает получать сообщения для окон и отправляет их соответствующей процедуре приема оконных сообщений. По сути, именно теперь ваша программа начинает выполняться в среде Windows. Интересно отметить, что значение возврата не поддерживается системными средствами Windows, хотя это могло принести пользу при отладке (вы могли бы отобразить это значение по завершении программы).

Создание экранной заставки

Экранная заставка — это форма, которая используется в течение короткого периода времени, после чего удаляется из памяти, чтобы не занимать ее пространство. Как отмечалось выше, экранная заставка должна появляться на время, пока приложение готовится к активным (видимым) действиям, а затем, когда приложение абсолютно готово к работе, удаляться. Поэтому, если вы познакомились с приведенной выше схемой функции WinMain(), вам должно быть ясно, что мы собираемся установить заставку при первой удобной возможности (после оператора try), а затем удалить ее, прежде чем приложение "возьмет власть в свои руки" (перед оператором Application->Run()).

Поскольку экранная заставка — не более чем форма, в нее можно положить все, что угодно, — рисунки, информацию в виде короткого текста, текущее состояние регистрации приложения и пр. Мы же поместим в форму строку состояния, чтобы отображать процесс загрузки форм в память.

Снова начнем с создания приложения, выбрав сначала команду File⇨Close All, а затем команды — File⇨New Application и File⇨Save Project As (Файл⇨Сохранить проект как). Сохраните Unit1 под именем MainUnit.cpp, Project1 под именем SplashProject.bpr и переименуйте с помощью инспектора объектов Form1 в MainForm. Затем выберите команду File⇨New Form (Файл⇨Создать форму) и переименуйте ее в SplashForm, сохранив файл ее модуля под именем SplashUnit.cpp. Теперь установите для всех свойств BorderIcons формы SplashForm значения false и выполните команду File⇨Save All (Сохранить все).

Затем, чтобы приложению было чем "заняться", выберите команду File⇨New Form пять раз. Тем самым вы заложите в свое приложение пять новых пустых форм. Для сохранения работы выполните еще раз команду File⇨Save All, оставив все имена, присвоенные по умолчанию. Эти пять форм имитируют в нашем приложении загрузку потенциально возможных различных форм при его запуске.

Теперь можно настроить экранную заставку. Сверните пять пустых форм и поместите в форму SplashForm компонент индикации выполнения TProgressBar (расположенный во вкладке Win32 палитры компонентов), установив его свойство Max равным 6. Затем сверните остальные две формы (SplashForm и MainForm) и выберите из меню C++Builder команду Project⇨View Source (Проект⇨Программа). Теперь нужно изменить код функции WinMain(), чтобы она выглядела в соответствии с содержимым листинга 30.5. Обратите внимание на то, что новый код предваряется комментарием, а старый — показан в закомментированном виде.

Листинг 30.5. Функция `WinMain()`

```
Application->Initialize();
//Новый код.
SplashForm = new TSplashForm(static_cast<void*>(NULL));
//Создаем нашу Splash-форму (заставку).
// Новый код.
SplashForm->Show();//Отображаем Splash-форму.
// Новый код.
Application->ProcessMessages();
//Создаем форму Mainform.
Application->CreateForm(__classid(TMainForm), &MainForm);
// Новый код.- Следующие две строки.
SplashForm->ProgressBar1->StepBy(1);//Инкрементируем индикатор
                                    //выполнения на единицу.
sleep(2);//Вставляем паузу, чтобы замедлить процесс.
Application->CreateForm(__classid(TForm1), &Form1);//Создаем форму Form1.
SplashForm->ProgressBar1->StepBy(1);
sleep(2);
Application->CreateForm(__classid(TForm2), &Form2);
SplashForm->ProgressBar1->StepBy(1);
sleep(2);
Application->CreateForm(__classid(TForm3), &Form3);
SplashForm->ProgressBar1->StepBy(1);
sleep(2);
Application->CreateForm(__classid(TForm4), &Form4);
SplashForm->ProgressBar1->StepBy(1);
sleep(2);
Application->CreateForm(__classid(TForm5), &Form5);
sleep(2);
delete SplashForm;//Удаляем SplashForm и освобождаем память.
Application->Run();//Запускаем приложение.
```

Чтобы при запуске индикаторов выполнения внести дополнительный эффект (появлением слова "Loading"), достаточно после каждой строки

```
SplashForm->ProgressBar->StepBy(1);
```

добавить строку

```
SplashForm->Label1->Caption+=".";.
```

Наконец, добавьте в начало вашего файла проекта (`SplashProject.cpp`) строки `#include SplashUnit.h` и `#include <dos.h>`. Теперь можно выполнить команду File⇨Save All, скомпилировать и запустить проект.

На заметку

Из-за использования функции `sleep(int);` вы получите предупреждение, поскольку это устаревшая C-функция. При желании ее можно заменить компонентом `TTimer` или циклом `for`.

Теперь у вас будет вполне работоспособная экранная заставка с индикатором выполнения, благодаря которой пользователь будет знать, какая часть приложения загружена в память.

Предотвращение запуска нескольких экземпляров приложения

Сложность реальных приложений постоянно растет. Зачастую они находятся в жесткой зависимости от весьма ограниченных сетевых и системных ресурсов, которые не позволяют корректно работать нескольким экземплярам одного приложения. Возможность запуска нескольких экземпляров приложения может не только сбить с толку вашего оператора, но также привести к совершенно неоправданным расходам системных ресурсов и снижению производительности. К счастью, обнаружение предыдущего экземпляра приложения и предотвращение запуска пользователем следующего — относительно простая задача.

Решение

В 16-разрядной системе Windows можно было бы просто проанализировать первые два параметра, передаваемые функции `WinMain()`. Они определены как `hCurInstance` и `hPrevInstance`.

Параметр `HCurInstance` возвращает дескриптор текущего экземпляра приложения, а параметр `hPrevInstance` — дескриптор предыдущего. Если в данный момент работает первый экземпляр приложения, параметр `hPrevInstance` возвращает значение `NULL`. К сожалению, в 32-разрядной среде Windows `hPrevInstance` всегда возвращает значение `NULL`, а посему от этого параметра нам нет никакого проку. Он присутствует здесь лишь из соображений обратной совместимости с 16-разрядным "наследством". Остается только гадать, почему инженеры компании Microsoft решили ликвидировать в 32-разрядной версии Windows этот источник информации.

Теперь мы должны "своими силами" определить, запущен ли уже экземпляр приложения. Существует множество различных способов выполнить эту задачу. Основной — оставлять при запуске приложения "след", т.е. своего рода индикатор работающего приложения, и удалять его по завершении работы.

При каждом запуске приложение должно проверять наличие этого индикатора. Если индикатор обнаружен, приложение соответствующим образом обрабатывает эту ситуацию и закрывается. В противном случае приложение сначала создает индикатор, после чего нормально приступает к работе. Завершая работу, оно должно удалить индикатор.

Имейте в виду, если приложению что-либо помешает нормально завершиться и удалить индикатор, вы не сможете снова запустить приложение до тех пор, пока не удалите индикатор вручную. В идеальном варианте хотелось бы использовать метод, подконтрольный операционной системе, и тогда, в случае скоропостижной "смерти" приложения, операционная система сама позаботилась бы об удалении всех следов его экземпляра, чтобы ничто не мешало нам запустить в работу новый экземпляр.

Так неужели такого метода не существует? Ну конечно же, он есть, и называется этот объект синхронизации Win32 мьютексом.

Слово *мьютекс* (mutex) представляет собой сокращение от слов *mutually exclusive* (взаимно исключающий). Мьютекс используется для ограничения доступа к системным ресурсам для одного потока выполнения. В нашем случае мы попытаемся ограничить "аппетиты" системы одним экземпляром приложения.

Программа

Выполните следующую последовательность действий, чтобы создать приложение SingleInstance. Если вы не желаете заниматься вводом текста программы, можете воспользоваться программой, приведенной на компакт-диске, прилагаемом к этой книге, хотя лично я

рекомендую вам создать приложение самостоятельно — это поможет лучше понять заложенные в нем идеи.

1. Сохраните приложение, щелкнув на кнопке с изображением стопки дискет, расположенной на панели инструментов C++Builder. Оставьте для модуля имя, присвоенное ему по умолчанию, но файлу проекта дайте имя `SingleInstance`.
2. Установите свойство `Caption` главной формы равным `The Only One!`.
3. Щелкните на кнопке View Unit (Просмотр модуля), расположенной на панели инструментов, и выберите из списка модуль `SingleInstance`.
4. Модифицируйте функцию `WinMain()` в модуле `SingleInstance`, как показано в листинге 30.6.

Листинг 30.6. Основной метод модуля `SingleInstance`

```
WINAPI WinMain(HINSTANCE, HINSTANCE, LPSTR, int)
{
    try
    {
        // Проверяем существование мьютекса.
        HANDLE Mutex = OpenMutex(MUTEX_ALL_ACCESS, false, "OneInstanceAllowed");
        if (Mutex == NULL) // Это - единственный экземпляр.
        {
           //Создаем мьютекс...
           Mutex = CreateMutex(NULL, true, "OneInstanceAllowed");
         }
         else // Это - не единственный экземпляр.
        {
          // Посылаем сообщение предыдущему экземпляру приложения
          // с просьбой восстановить и показать себя.
          SendMessage(HWND_BROADCAST, RegisterWindowMessage("OnlyOne"), 0, 0);
            return 0;
          }
          // Позволяем приложению нормально работать...
        Application->Initialize();
        Application->CreateForm(__classid(TForm1), &Form1);
        Application->Run();

        // Освобождаем мьютекс...
        ReleaseMutex(Mutex);
    }
        catch (Exception &exception)
     {
          Application->ShowException(&exception);
     }
     return 0;
}
```

Откройте файл `MainUnit.h`, щелкнув правой кнопкой мыши на вкладке `Unit1.cpp` в редакторе программ и выберите из контекстного меню команду Open Source/Header File (Открыть исходный/заголовочный файл).

Добавьте в закрытый (private) раздел переменную для хранения нашего "оконного" сообщения и прототип функции для обработчика этого сообщения, как показано в листинге 30.7.

Листинг 30.7. Объявление класса Form1

```
class TForm1 :public TForm
{
    published: // Компоненты, управляемые IDE.
        void __fastcall FormCreate(TObject *Sender);
        void __fastcall FormDestroy(TObject *Sender);
    private: // Объявления пользователя.
        unsigned int msgRestore;
        bool __fastcall MessageHandler(TMessage &Message);
    public: // Объявления пользователя.
        fastcall TForm1(TComponent* Owner);
};
```

Выберите в окне инспектора объектов Form1, затем — вкладку Events и дважды щелкните на событиях OnCreate и OnDestroy, чтобы создать обработчики этих событий.

Введите код в методы FormCreate() и FormDestroy() и создайте функцию MessageHandler, как показано в листинге 30.8.

Листинг 30.8. Form1.cpp

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

#include "Unit1.h"
//---------------------------------------------------------------
#pragma package(smart_init)
#pragma resource "*.dfm"
TForm1 *Form1;
//---------------------------------------------------------------
__fastcall TForm1::TForm1(TComponent*Owner)
        :TForm(Owner)
{
}
//---------------------------------------------------------------

bool __fastcall TForm1::MessageHandler(TMessage &Message)
{
     if (Message.Msg == msgRestore)
     {
          Application->Restore();
          Application->BringToFront();
          return true;
     }
     else return false;
}
```

```
void __fastcall TForm1::FormCreate(TObject *Sender)
{
     msgRestore = RegisterWindowMessage("OnlyOne");
     Application->HookMainWindow(MessageHandler);
}
//---------------------------------------------------------------

void __fastcall TForm1::FormDestroy(TObject *Sender)
{
     Application->UnhookMainWindow(MessageHandler);
}
//---------------------------------------------------------------
```

Скомпилируйте и постройте приложение, сверните окно C++Builder, перейдите в папку, где сохранен проект, и выполните приложение. В данном конкретном случае приложение необходимо запускать вне IDE C++Builder, чтобы можно было увидеть, что происходит при попытке выполнить более одного экземпляра.

`RegisterWindowMessage()` — это функция Win32 API, которая регистрирует строку с гарантированно глобально уникальным идентификатором сообщения. При каждом вызове затрачиваются некоторые системные ресурсы, что и заставило нас создать переменную `msgRestore`. Мы можем просто вызывать функцию `RegisterWindowMessage("OnlyOne")` каждый раз, когда вызывается обработчик пользовательского сообщения, но поскольку он может быть вызван тысячи раз за очень короткий промежуток времени, эффективность такого варианта была бы очень низкой. Поэтому мы обращаемся к ней только один раз при создании формы и сохраняем результат в переменной, с которой очень быстро и эффективно можно сравнивать приходящее сообщение.

Метод `Application->HookMainWindow()` позволяет "сцепить" обработчик пользовательского сообщения с главным окном приложения. В данном случае мы используем его для анализа всех оконных сообщений, чтобы отыскать среди них то, которое несет в себе приказ восстановить приложение и вывести его на передний план. Метод `Application->UnhookMainWindow()` попросту аннулирует это сцепление, хотя в действительности не является обязательным для приложений с одной формой (как в данном случае). Если этот метод используется для сцепления с главным окном приложения некоторого диалогового окна или одной формы в приложении с несколькими формами, то при разрушении этого диалогового окна главное окно необходимо "отцепить", в противном случае уничтожаемое диалоговое окно "погубит" и главное окно приложения, что вызовет неожиданный, "скоропостижный" конец всего приложения. В нашем случае, закрывая главную форму, мы сознательно ожидаем наступления конца приложения, да и Бог с ним.

При инициализации приложения оно опрашивает систему на наличие мьютекса с именем `OneInstanceAllowed`. Если таковой существует, приложение передает операционной системе оконное сообщение `OnlyOne`. В уже работающем экземпляре нашего приложения это сообщение перехватывается обработчиком (как и все оконные сообщения), в ответ на которое приложение будет восстановлено и перейдет на передний план. Если мьютекс не существует, приложение создаст его и продолжит свою работу в обычном режиме.

Еще один способ заставить первый экземпляр приложения переместиться на передний план — использовать функцию `FindWindow()`. В качестве параметров она принимает имя класса окна главной формы приложения и настоящее имя окна. Если вместо настоящего имени окна передать значение `NULL`, это будет означать соответствие всем окнам заданного класса, невзирая на их имена. В этом примере естественно предположить, что в системе есть только одна активная форма `TForm1`. Однако на практике следует присваивать уникальное имя каждому классу формы.

Если функции FindWindow() не удается вернуть действительный дескриптор, можно допустить наличие ошибки. В противном случае заставляем это окно показать себя "во всей красе" с помощью функции ShowWindow(), а главное окно приложения — переместиться на передний план с помощью функции SetForegroundWindow(). Листинг 30.9 содержит функцию WinMain(), модифицированную для использования метода FindWindow(), осуществляющего перемещение экземпляра приложения на передний план.

Листинг 30.9. `Form1.cpp`

```
WINAPI WinMain(HINSTANCE,HINSTANCE,LPSTR,int)
{
    try
    {
        // Проверяем существование мьютекса.
        HANDLE Mutex = OpenMutex(MUTEX_ALL_ACCESS, false, "OneInstanceAllowed");
          if (Mutex == NULL) // Это единственный экземпляр.
        {
            // Создаем мьютекс...
            Mutex = CreateMutex(NULL, true, "OneInstanceAllowed");
        }
        else // Это не единственный экземпляр.
        {
            HWND handle = FindWindow("TForm1", NULL);
            if (!handle) Application->MessageBox(
                    "Previous application instance not found.",
                    "Application Error!", MB_OK);
            else
            {
                ShowWindow(handle, SW_SHOWNORMAL);
                SetForegroundWindow(handle);
            }
        }

        // Пусть приложение выполняется нормально...
        Application->Initialize();
        Application->CreateForm(__classid(TForm1), &Form1);
        Application->Run();

        // Освобождаем мьютекс...
        ReleaseMutex(Mutex);
    }
        catch (Exception &exception)
    {
        Application->ShowException(&exception);
    }
    return 0;
}
```

По возвращении приложения из метода Application->Run() мьютекс освобождается, и приложение красиво завершается.

Заключение

Мьютексы — простой и эффективный способ организации приложений, гарантирующий, что пользователи не смогут запустить в работу более одного экземпляра приложения.

Перемещая (с помощью функции `FindWindow()`) предыдущий экземпляр приложения на передний план, мы создаем у пользователя впечатление, что приложение запустилось нормально, а на самом деле предотвращаем случайное выполнение нескольких экземпляров.

Создание обработчика пользовательских сообщений — задача более сложная, но благодаря этому мы без особых проблем организуем общение непосредственно с другими экземплярами приложения или даже другими приложениями. Мне нравится этот метод своей элегантностью и эффективностью.

Организация инструмента „перетащить и опустить“

Инструмент “перетащить и опустить” (drag-and-drop) — одно из старейших средств операционной системы Microsoft Windows. Перетаскивание экранных объектов с помощью мыши стало естественным расширением поведения человека. Эта функция мыши легко усваивается новыми пользователями Windows, поэтому ее имеет смысл реализовать во всех пользовательских приложениях (где, безусловно, это уместно). К счастью, идея этого инструмента весьма проста, да и ее реализация в C++Builder не вызывает особых трудностей.

Решение

Для реализации в приложении инструмента “перетащить и опустить” необходимо сначала проинформировать операционную систему о том, что приложение готово принимать файлы, “перетаскиваемые” и “опускаемые” с помощью мыши. Это можно сделать посредством метода `DragAcceptFiles()` интерфейса Win32 API. Затем нужно обработать события, возникающие в результате действия “опустить”. Это реализуется путем отображения сообщений и обработчика событий для сообщения `WM_DROPFILES`, который будет читать имя перетаскиваемого файла и действовать соответствующим образом.

Описание программы

Для иллюстрации этой идеи мы не будем строить средство просмотра факс-материалов. Усложнение задачи не означает улучшение демонстрации механизма действия инструмента “перетащить и опустить”. Поэтому вместо средства просмотра факс-материалов мы создадим приложение, которое очень напоминает редактор конфигурации системы (System Configuration Editor), поставляемый с Windows. Чтобы увидеть его в работе, щелкните на кнопке Start (Пуск), выберите из меню команду Run (Выполнить), введите в диалоговом окне Run (Запуск программы) текст `Sysedit` и щелкните на кнопке OK. Немного поэкспериментировав, можно заметить, что приложение `Sysedit` не обрабатывает перетаскиваемые файлы. Из соображений краткости наше небольшое приложение не будет выполнять другие функции приложения `Sysedit`, включая возможность сохранять отредактированные файлы. При желании эту функцию вы сможете легко реализовать сами. Обратите внимание, что на рис. 30.4 приложение `DragDrop` отображает некоторые `readme`-файлы, которые входят в поставку C++Builder 5.

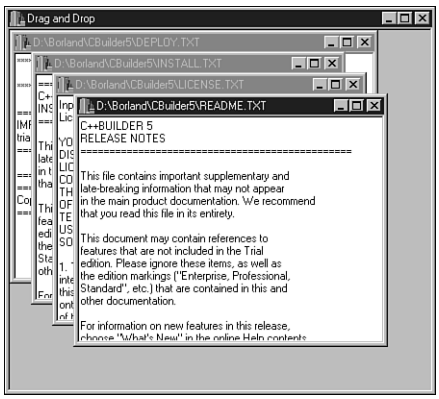


Рис. 30.4. Приложение DragDrop в действии

Не редактируйте содержимое никаких дочерних окон в программе System Configuration Editor, если не уверены в своих действиях. В зависимости от версии Windows, эти файлы сообщают операционной системе, как должен быть осуществлен ее запуск. Появление одной ошибки в этих файлах или ввод некорректных значений может привести к возникновению больших проблем.

Если вы открыли редактор System Configuration Editor из любопытства, закройте его, предотвратив возможность работы с ним. Выполните следующие действия для создания приложения DragDrop или просто загрузите его с компакт-диска, прилагаемого к данной книге.

1. Запустите C++Builder и создайте новое приложение.
2. Переименуйте форму Form1 в MainForm.
3. Создайте новую форму с именем ChildForm. Установите ее свойства ClientHeight и ClientWidth равными приблизительно значениям 250 и 350, соответственно.
4. Добавьте в форму компонент TRichEdit из вкладки Win32 палитры компонентов и установите его свойству Align значение alClient.
5. Сохраните приложение, щелкнув на кнопке с изображением стопки гибких дисков, расположенной на панели инструментов C++Builder. Сохраните модуль главной формы под именем MainUnit.cpp, модуль дочерней формы ChildForm — под именем ChildUnit.cpp, а файл проекта — под именем DragDrop.bpr.

После создания формы можно ввести код в обработчики событий.

Чтобы проинформировать операционную систему о том, что вы хотели бы принимать перетаскиваемые файлы, необходимо вызвать метод DragAcceptFiles(). Лучше всего это сделать в обработчике события OnCreate для главной формы. Выберите из раскрывающегося списка в верхней части окна инспектора объектов элемент MainForm, перейдите во вкладку Events, дважды щелкните на событии OnCreate и вставьте следующую строку кода в обработчик конечного события.

```
DragAcceptFiles(Handle, True);
```

Чтобы создать обработчик события DragDrop, откройте заголовочный файл для главной формы, щелкнув правой кнопкой мыши на вкладке MainUnit.cpp в окне редактора программ,

и выберите из контекстного меню команду Open Header/Source File. Вставьте следующий текст программы в открытый (public) раздел объявления класса TMainForm:

```
class TMainForm :public Tform
{
__published:  // Компоненты, управляемые IDE.
    void __fastcall FormCreate(TObject *Sender);
private:      // Объявления пользователя.
    void virtual __fastcall WMDropFiles(TWMDropFiles &message);
public:       // Объявления пользователя.
    __fastcall TMainForm(TComponent *Owner);
    BEGIN_MESSAGE_MAP
    MESSAGE_HANDLER(WM_DROPFILES, TWMDropFiles, WMDropFiles)
    END_MESSAGE_MAP(TForm);
};
```

Теперь можно снова перейти в файл MainUnit.cpp и добавить в его конец текст программы, приведенный в листинге 30.10. Советуем оставить все комментарии.

Листинг 30.10. Обработчик события WMDropFiles

```
// Событие генерируется, когда файл (файлы) "опускаются" в приложение.
void __fastcall TMainForm::WMDropFiles(TWMDropFiles &message)
{
    AnsiString FileName;
    FileName.SetLength(MAX_PATH);

    int Count = DragQueryFile((HDROP)message.Drop, 0xFFFFFFFF, NULL, MAX_PATH);

    // Опрашиваем с помощью индекса файлы операционной системы
    // на предмет их имен...
    for (int index =0;index <Count;++index)
    {
        // Читаем имя (FileName) перетаскиваемого файла. Из
        // следующего оператора это трудно понять, но тем не
        // менее это так. Как вы думаете, почему Delphi и
        // C++Builder (несмотря ни на что) столь популярны?
        // Обратитесь к справочному файлу Windows API Win32.hlp и
        // изучите все, что относится к DragQueryFile.
        FileName.SetLength(
             DragQueryFile((HDROP)message.Drop, index,FileName.c_
↳str(), MAX_PATH));

        // Проверяем расширение в имени файла.
        // Если это текстовый или RTF-файл (Rich Text file), то...
        if (UpperCase(ExtractFileExt(FileName)) == ".TXT" ||
↳UpperCase(ExtractFileExt(FileName)) == ".RTF")
         {
             // Создаем новую дочернюю форму...
           TChildForm *Viewer = new TChildForm(Application);
           // Отображаем файл...
           Viewer->Caption = FileName;
```

```
            Viewer->RichEdit1->Lines->LoadFromFile(FileName);
            Viewer->Show();
        }
    }
    // Уведомляем операционную систему о завершении...
    DragFinish((HDROP) message.Drop);
}
```

Чтобы предотвратить утечку памяти, необходимо гарантировать ее корректное освобождение при каждом закрытии средства просмотра. Выберите форму `ChildForm` в окне Object Inspector. Перейдите во вкладку Events и дважды щелкните на событии `OnClose`, чтобы создать обработчик события `OnClose`. Введите в этот обработчик события следующий код.

```
Action = caFree;
```

Откройте в редакторе программ файл `MainForm.cpp`, выберите команду File⇨Include Unit Header (Файл⇨Включить заголовок модуля) и выберите дочернюю форму. Это заставит главную форму `MainForm` учитывать существование `ChildForm`, в результате чего компилятор будет "знать", что мы имеем в виду, ссылаясь на дочернюю форму.

Скомпилируйте и запустите приложение. При перетаскивании обычного текстового или RTF-файла в это приложение в его клиентской области будет открываться средство просмотра текста.

Как работает приложение

Когда приложение инициализирует и создает главную форму `MainForm`, оно вызывает Win32 API-метод `DragAcceptFiles()`, передавая ему дескриптор приложения и значение `true`, означающее для операционной системы, что приложение готово принимать "перетаскиваемые и опускаемые" файлы. Передача операционной системе значения `false` запрещает выполнение операций "перетащить и опустить" в приложении.

Если эта операция разрешена, приложение будет принимать от Windows сообщение `WM_DROPFILES` для каждого получаемого им файла. Для надлежащей обработки этих сообщений необходимо определить макрос `MESSAGE_HANDLER`, который представляет собой структуру, связывающую конкретное сообщение Windows с одним из обработчиков пользовательских сообщений приложения. Отображение сообщений, определенное в приложении `DragDrop`, связывает сообщение `WM_DROPFILES` с обработчиком сообщений `WMDropFiles`.

В обработчике сообщений `WMDropFiles` с помощью метода `DragQueryFile()` у операционной системы будет запрашиваться информация о перетаскиваемых файлах. Ниже приводится определение Microsoft метода `DragQueryFile()`. Его можно найти в справочном файле `Win32.hlp`.

Функция `DragQueryFile()` считывает имена перетаскиваемых файлов.

```
UINT DragQueryFile(
   HDROP hDrop, //Дескриптор структуры для перетаскиваемых файлов
   UINT iFile, // Индекс файла для запроса.
   LPTSTR lpszFile, // Буфер для возвращаемого имени файла.
   UINT cch // Размер буфера для имени файла.
);
```

Параметры.

- `hDrop`. Идентифицирует структуру, содержащую имена перетаскиваемых файлов.

- iFile. Задает индекс файла для запроса. Если значение параметра iFile равно 0xFFFFFFFF, функция DragQueryFile() возвращает количество файлов, которые подверглись операции "перетащить и опустить". Если значение параметра iFile находится между нулем и общим значением таких файлов, функция DragQueryFile() копирует имя файла с соответствующим значением в буфер, адресуемый параметром lpszFile.
- lpszFile. Указывает на буфер для приема имени перетаскиваемого файла при возврате функции. Имя файла является строкой с завершающим нулевым символом. Если этот параметр равен значению NULL, функция DragQueryFile() возвращает запрашиваемый размер буфера в символах.
- cch. Задает размер буфера lpszFile в символах.

Возвращаемые значения.

Если функция копирует имя файла в буфер, значение возврата представляет собой количество скопированных символов, не считая завершающий нулевой символ. Если значение индекса равно 0xFFFFFFFF, значение возврата представляет собой количество перетаскиваемых файлов. Если значение индекса лежит между нулем и общим значением таких файлов и при этом адрес буфера lpszFile характеризуется значением NULL, значение возврата равно запрашиваемому размеру буфера в символах (без учета завершающего нулевого символа).

Когда приложение DragDrop принимает "перетаскиваемый" файл, оно выполняет метод WMDropFiles(), который использует дескриптор Message для запроса у операционной системы количества "перетаскиваемых" файлов. Затем опрашивается список файлов, и каждое расширение проверяется на предмет совпадения с .txt или .rtf. Если расширение файла совпадает с одним из этих двух вариантов, приложение создает экземпляр ChildForm и загружает файл в компонент TRichText для отображения пользователю. При закрытии каждого дочернего окна ChildWindow вызывается действие caFree, которое освобождает память, связанную с экземпляром формы ChildForm.

Резюме по операции „перетащить и опустить"

Несмотря на то что не существует стандартной VCL-оболочки для операции "перетащить и опустить", C++Builder позволяет легко ее реализовать в любом приложении. Если вы не использовали эту возможность до сих пор, считая ее реализацию слишком трудной задачей, то теперь вы убедились, что она решается достаточно просто.

Перехват экрана

Вероятно, вам доводилось иметь дело с приложениями, которые позволяют захватывать целый рабочий стол Windows, активное окно или даже некоторую область экрана. В этом разделе мы покажем, насколько просто, используя C++Builder, создать приложение, способное выполнять подобные действия.

Как Windows работает с окнами

В этом разделе мы рассмотрим, как Windows отображает различные типы окон, а также факторы, влияющие на видимость окон на экране.

Windows имеет стандартное окно Desktop (Рабочий стол), которое отображается постоянно. Окно рабочего стола автоматически создается при каждом запуске Windows. Оно служит в качестве базового окна для всех окон всех приложений. Для раскраски фона экрана ок-

но рабочего стола использует растровый файл. Для получения дескриптора окна Desktop используйте функцию GetDesktopWindow().

Windows отображает окно, если его стиль имеет значение WS_VISIBLE; в противном случае это окно сокрыто. В среде C++Builder можно скрывать или отображать окно, изменяя соответствующее свойство. Функция ISWindowVisible() позволяет определить, видимо ли в данный момент окно.

Установка стиля WS_VISIBLE для окна вовсе не означает, что пользователь будет видеть его на экране. Оно может быть полностью закрыто другими окнами или попасть за пределы экрана.

Все отображаемые приложениями окна перекрывают друг друга в соответствии с z-порядком (z-последовательностью), который определяется в электронной справочной системе как "умозрительная удаленность объекта от поверхности экрана". Закрывают ли элемент управления другие элементы управления — зависит от его z-порядка по отношению к этим элементам управления. Окно с наименьшим значением координаты по оси z перекрывается всеми другими окнами, а окно с наибольшим значением координаты по оси z, наоборот, само перекрывает все другие окна.

Когда пользователь активизирует одно из окон, Windows поддерживает позицию каждого окна в z-последовательности. Активизированное окно будет располагаться на вершине этой z-последовательности. Для выявления "самого высокого" окна в z-последовательности можно использовать функцию GetTopWindow(), а затем вернуть дескриптор этого окна.

Windows также поддерживает позицию каждого окна в z-последовательности, в соответствии с его типом. Например, самые важные окна всегда находятся на вершине z-последовательности. Следовательно, самые важные окна будут перекрывать остальные, "менее важные" окна. Важное окно имеет стиль WS_EX_TOPMOST. Для проверки окна на "значимость" можно использовать функцию GetWindowLong().

Решение

На компакт-диске, прилагаемом к этой книге, мы создали проект ScrnCapture.bpr, который позволит пользователю захватить либо весь экран, либо выделенную его часть, либо активное видимое окно.

Чтобы воссоздать программу ScrnCapture, создайте новое приложение, поместите в форму компоненты и установите их свойства, как показано в табл. 30.6.

Таблица 30.6. Компоненты и их свойства для программы ScrnCapture

Компонент	Свойство	Значения
TForm	Name	Form1
	Caption	Screen Capture
TMainMenu	Name	MainMenu1
	MenuItem	&File
	MenuItem	Save &As
	MenuItem	E&xit
	MenuItem	&Action
	MenuItem	Capture &Desktop Image
	MenuItem	Capture &Active Window
	MenuItem	Capture &Region

Окончание табл. 30.6

Компонент	Свойство	Значения
Timage	Name	Image1
TScrollBox	Name	ScrollBox1
	Align	alClient
TSaveDialog	Name	SaveDialog1
	Filter	BMP (стандартный формат графических файлов Windows)

Назначение формы Form1 — служить в качестве интерфейса. Основные элементы этой формы — три элемента (MenuItem) меню Action (компонента TMainMenu). Эти три элемента на самом деле являются ядром проекта, поэтому они подробно описываются в следующих трех разделах.

Чтобы завершить разработку проекта, добавьте в него новую форму, назовите ее Form2 и установите ее свойству FormStyle значение fsStayOnTop, а свойству WindowState — значение wsMaximized. Затем поместите в форму компонент TImage и установите его свойству Align значение alClient. Форма Form2 будет использована только в третьем элементе меню Action, где она заменит рисунок рабочего стола, чтобы пользователь мог выделить область захвата.

Захват целого экрана

В файле Unit1.cpp проекта ScrnCapture нетрудно найти следующую функцию, которая копирует рабочий стол в изображение приемника:

```
void __fastcall TForm1::CaptureDesktop(HDC DesImage)
{
    HDC hdc = GetDC(0);
    BitBlt(DesImage,
               0, 0, Screen->Width, Screen->Height,
               hdc, 0, 0, SRCCOPY);
    ReleaseDC(0, hdc);
}
```

Функция CaptureDesktop() принимает в качестве параметра дескриптор контекста устройства клиентской области приемника изображения. Сначала она считывает дескриптор контекста устройства рабочего стола, а затем копирует изображение рабочего стола в изображение приемника DesImage с помощью функции BitBlt().

Захват активного окна

Для захвата активного окна, которое закрывает собой все другие окна, необходимо выполнить следующие действия.

1. Воспользуйтесь функцией GetTopWndHandle(), чтобы определить, какое окно является активным и закрывает все другие окна, а затем считайте его дескриптор.
2. Определите размер ограничительного прямоугольника этого окна.
3. Выполните захват изображения с экрана (которое определяется прямоугольником окна) с помощью функции CaptureWndImage().

Функция GetTopWndHandle(), как показано в листинге 30.11, считывает дескриптор активного окна. Сначала она получает дескриптор верхнего окна, принадлежащего рабочему

столу, а затем последовательно опрашивает все окна, чтобы считать дескриптор активного окна. Активное окно должно быть видимо и иметь значение стиля WS_EX_TOPMOST.

Листинг 30.11. Функция GetTopWndHandle()

```
HWND __fastcall TForm1::GetTopWndHandle(void)
{
    HWND hwnd=GetTopWindow(0);
    // Не выходим из цикла, пока не найдем активное окно.
    while (hwnd != 0)
    {
        if (IsWindowVisible(hwnd)&&(GetWindowLong(hwnd,
                              GWL_EXSTYLE) & WS_EX_TOPMOST )==0)
        {
            // Лишь в случае, когда все окна свернуты, получаем
            // окно рабочего стола (desktop).
            if (IsIconic(hwnd))hwnd=GetDesktopWindow();
                return hwnd;
        }
        hwnd=GetNextWindow(hwnd,GW_HWNDNEXT);
    }
    // Просто, чтобы заблокировать предупреждение (warning)...
    return 0;
}
```

Захват выделенных частей экрана

Для захвата выделенных частей экрана в программе ScrnCapture используются следующая последовательность действий.

1. Получаем полное изображение рабочего стола с помощью функции CaptureDesktop().
2. Копируем изображение рабочего стола во временный компонент TImage, который размещается на другой форме (Form2). Эта форма будет полностью закрывать собой рабочий стол.
3. Разрешаем пользователю выделить область, которая будет скопирована. Чтобы выделить эту область, пользователь может просто щелкнуть левой кнопкой мыши (для определения первого угла области), а затем, удерживая эту кнопку, перетащить указатель мыши в позицию второго угла области и там отпустить кнопку мыши.
4. Копируем выделенную область в компонент приемника TImage с помощью функции TForm2::CaptureImage().
5. Закрываем временную форму.

Код, представленный в листинге 30.12, используется, чтобы разрешить пользователю выделить область, которую он хочет скопировать.

Листинг 30.12. Выделение области копирования

```
void __fastcall TForm2::Image1MouseDown(TObject *Sender,
            TMouseButton Button, TShiftState Shift, int X, int Y)
{
    IsRegionSelected = true;
```

```
    Origin = Point(X,Y);
    MovePt = Origin;
}
//---------------------------------------------------------------
void __fastcall TForm2::Image1MouseMove(TObject *Sender,
                                TShiftState Shift, int X,int Y)
{
    if (IsRegionSelected){
        DrawRect(Origin, MovePt, pmNotXor);
        MovePt = Point(X,Y);
        MovePt.x = MovePt.x + Image1->Left;
        MovePt.y = MovePt.y + Image1->Top;
        DrawRect(Origin, MovePt, pmNotXor);
    }
}
//---------------------------------------------------------------
void __fastcall TForm2::Image1MouseUp(TObject *Sender,
           TMouseButton Button, TShiftState Shift, int X, int Y)
{
    if (IsRegionSelected)
    {
        MovePt = Point(X,Y);
        IsRegionSelected = false;
        CaptureRegion();
    }
    // Возврат
    Form1->Visible = true;
    Form2->Visible = false;
}
//---------------------------------------------------------------
void __fastcall TForm2::DrawRect(TPoint TopLeft,
                            TPoint BottomRight, TPenMode KAMode)
{
    Canvas->Pen->Mode = KAMode;
    Canvas->Pen->Width = 2;
    Canvas->Rectangle(TopLeft.x, TopLeft.y,
                                  BottomRight.x, BottomRight.y);
}
```

Функция `CaptureRegion()` приведена в листинге 30.13.

Листинг 30.13. Функция `CaptureRegion()`

```
void __fastcall TForm2::CaptureRegion(void)
{
    if((MovePt.y-Origin.y) != 0)
    {
        int KAWidth = abs(MovePt.x-Origin.x);
        int KAHeight = abs(MovePt.y-Origin.y);
        int Orgx = Origin.x, Orgy = Origin.y;
        if (MovePt.y < Origin.y) Orgy = MovePt.y;
```

```
        if (MovePt.x < Origin.x) Orgx = MovePt.x;
        // Копируем выделенную часть рабочего стола в
        // изображение формы Form1.
        StretchBlt(Form1->Image1->Canvas->Handle,
                   0, 0,
                   KAWidth,
                   KAHeight,
                   Image1->Canvas->Handle,
                   Orgx, Orgy,
                   KAWidth,
                   KAHeight,
                   SRCCOPY);
        // Настраиваем размеры изображения формы Form1.
        Form1->Image1->Width = abs(MovePt.x-Origin.x);
        Form1->Image1->Height = abs(MovePt.y-Origin.y);
    }
    else
    {
        Form1->Image1->Picture->Bitmap->Canvas->Brush->Color =
                                                    clWhite;
        Form1->Image1->Picture->Bitmap->Canvas->FillRect (
                                              Canvas->ClipRect);
    }
}
```

Вы можете выполнить либо программу `ScrnCapture.exe`, которая находится на прилагаемом к книге компакт-диске, либо запустить это приложение, если у вас открыт проект ScrnCapture. На рис. 30.5 эта программа показана в действии.

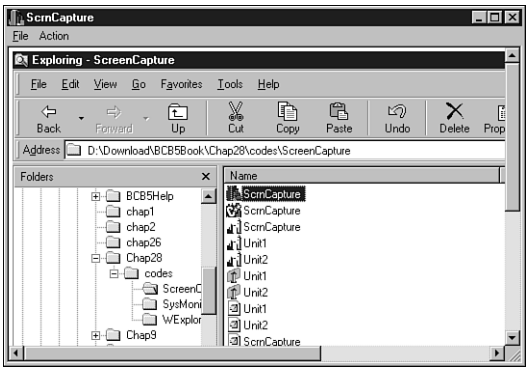


Рис. 30.5. Программа ScrnCapture в действии

Резюме по захвату экрана

Программу ScrnCapture можно усовершенствовать, заставив ее, например, захватывать активное контекстное меню или клиентскую область активного окна. Кроме того, для выполнения этого захвата можно назначить комбинацию клавиш.

Реализация компонента TJoyStick

В книге *The Borland C++Builder How-To* достаточно хорошо подан материал, посвященный API-мультимедиа, а именно связанный с использованием джойстика, благодаря чему значительно облегчается его реализация. Однако было бы еще лучше, если бы для этих целей существовал простой компонент. Это существенно сэкономило бы время при разработке игр.

Прежде всего нам потребуется пакет, который представляет собой DLL-библиотеку особого назначения, содержащую этот компонент. Закройте любые открытые проекты и выберите из меню C++Builder команду File⇨New, а затем из открывшегося диалогового окна — команду New⇨Package.

C++Builder сгенерирует пакет, показанный на рис. 30.6.

Новому проекту по умолчанию присваивается имя `Package1.cpp`, которое будет ему принадлежать до тех пор, пока вы не сохраните его с новым именем. Заметьте, что имя проекта не должно совпадать с именем компонента, который вы собираетесь создать и разместить в этом пакете. С другой стороны, имя должно соответствовать создаваемому компоненту. Давайте назовем наш пакет `Joystick` или, может быть, `MyPackage`, если вы собираетесь добавлять другие компоненты. Как показано в листинге 30.14, созданный `.cpp`-файл легко распознается как DLL-оболочка. Пока что выполните команду File⇨Save As и при сохранении присвойте своему новому пакету подходящее имя.

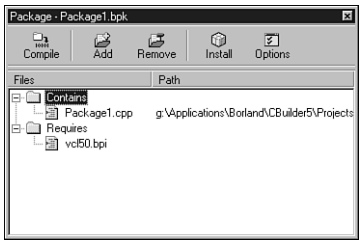


Рис. 30.6. Окно создания пакета C++Builder 5

Не забудьте сохранить *пакет*, а не *проект*. Для этого выделите этот пакет и выполните команду File⇨Save As.

Листинг 30.14. DLL-оболочка, используемая для хранения VCL-компонентов

```
//---------------------------------------------------------------
#include <vcl.h>
#pragma hdrstop
USERES("Package1.res");
USEPACKAGE("vcl40.bpi");
//---------------------------------------------------------------
#pragma package(smart_init)
//---------------------------------------------------------------
//Исходный пакет.
//---------------------------------------------------------------
int WINAPI DllEntryPoint(HINSTANCE hinst, unsigned long reason, void*)
{
    return 1;
}
//---------------------------------------------------------------
```

Продолжайте работу, сохранив пакет с придуманным вами именем. Теперь компонент можно поместить в проект. Из главного меню C++Builder выберите команду New⇨Component

(Создать⇨Компонент). Откроется диалоговое окно, запрашивающее информацию о новом компоненте.

Первый обращенный к вам вопрос при создании нового компонента касается его предка (рис. 30.7).

Раскрыв список Ancestor Type (Тип предка), нетрудно заметить, что в нем перечислены не все типы объектов. Например, нельзя "вывести потомство" из типа `AnsiString`. Тем не менее у вас есть еще немало шансов попасть впросак, сделав неправильный выбор. Лучше всего выбрать такой элемент управления, который ближе всего к тому, который вы хотите создать. Например, если вы хотите создать многоязычную надпись, вам следует взять за основу компонент `TLabel`. Поскольку ни один из элементов списка не походит на джойстик, мы воспользуемся компонентом `TWinControl`.

Затем нужно присвоить компоненту имя класса. Например, вполне подойдет имя `TJoyStick`. Если для имени класса в поле Unit File Name (Имя файла модуля) выбрать вариант `T<Whatever>`, C++Builder предложит использовать путь `$(BCB)\Lib` и уберет из имени класса начальную букву `T` (если таковая присутствует). Чтобы предоставить исходный файл для вашего класса, будет использована остальная часть имени класса с расширением `.cpp`. Стоит отметить, что при открытом исходном файле компонента функциональная клавиша <F12> не выполняет никакого действия, поскольку не существует какой бы то ни было формы или другой визуальной информации для перехода. Кроме того, именно здесь видны ограничения, налагаемые на имена классов, компонентов, пакетов и т.д. Они заключаются в том, что просто не может быть двух исходных файлов с одинаковыми именами (файлов с расширением `.cpp`).

Переходя к следующему полю, необходимо решить, на какой вкладке будет размещен новый элемент управления. Один из вариантов — разместить его на собственной вкладке палитры компонентов. Можно было бы также поместить его во вкладку Samples (Примеры) или любую другую. Стоит, конечно, создать вкладку личных компонентов, которая бы пополнялась по мере надобности. В данном случае мы просто введем `TJoyStick`, как показано на рис. 30.8.

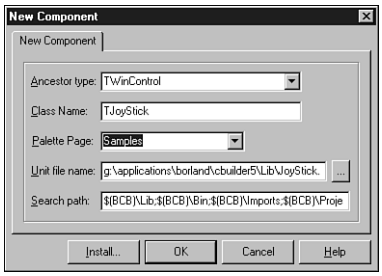


Рис. 30.7. Диалоговое окно New Component (Новый компонент)

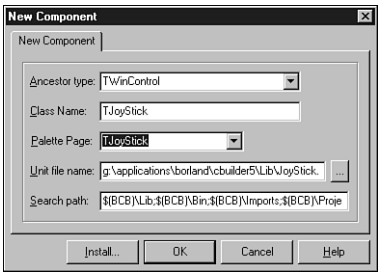


Рис. 30.8. Диалоговое окно New Component с выбранным элементом списка Palette Page

Код, который автоматически генерируется на основании выбранных вами вариантов, аналогичен содержимому листинга 30.15. Варианты этого кода, сгенерированные различными версиями C++Builder, могут несколько отличаться друг от друга.

Этот код включает три важные процедуры. Первая — осуществляет контроль недопущения определения чистых виртуальных методов. Вторая — выполняет инициализацию, а третья — регистрирует компонент. Теперь рассмотрим стандартное объявление класса, приведенное в заголовочном файле `JoyStick.h` для добавляемого нами класса (см. листинг 30.15).

Листинг 30.15. Файл `JoyStick.h`

```
//---------------------------------------------------------------------
#ifndef JoyStickH
#define JoyStickH
//---------------------------------------------------------------------
#include <SysUtils.hpp>
#include <Controls.hpp>
#include <Classes.hpp>
#include <Forms.hpp>
//---------------------------------------------------------------------
class PACKAGE TJoyStick : public TWinControl
{
private:
protected:
public:
     __fastcall TJoyStick(TComponent* Owner);
__published:
};
//---------------------------------------------------------------------
#endif
```

Первое, что вы должны были заметить в объявлении класса, это то, что класс определен как пакет (`PACKAGE`) и что он выведен из класса `TWinControl`. Подобно другим не-VCL-классам, именно здесь определяются методы вашего класса. Вы видите единственное определение метода, предназначенного для выполнения операций инициализации. Это пустое определение можно рассматривать как фундамент, на котором можно возводить остальные части сооружения. Но перед началом строительства следует сохранить этот модуль, выполнив команду File⇨Save или File⇨Save As. Теперь выберите команду View⇨Project (Вид⇨Проект), чтобы вернуться на уровень пакета; дважды щелкните на элементе `JoyStick.bpl`, чтобы открыть диалоговое окно Package (Пакет), и щелкните на кнопке Add (Добавить). Щелкните на кнопке Browse (Обзор) и выберите файл `TJoyStick.cpp`, чтобы добавить его в пакет. Ваш пакет должен иметь примерно такой вид, как на рис. 30.9.

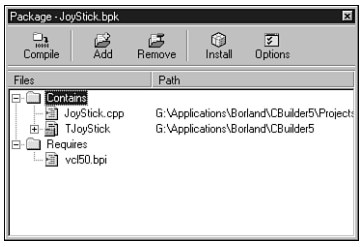


Рис. 30.9. Обновленное диалоговое окно Package

Вид вашего пакета будет несколько отличаться от того, что показано на рис. 30.9. Все дело в файле `TJoyStick.dcr`, который используется в данном примере. Этот `.dcr`-файл содержит растровое изображение, предназначенное для использования во вкладке Palette (Палитра). Он автоматически добавляется вместе с `.cpp`-файлом при добавлении последнего к пакету. `.dcr`-файл можно создать, используя средство Image Editor (Редактор изображений) из меню Tools (Инструменты). Создайте `.dcr`-файл с тем же именем, что и у вашего компонента, добавьте в него растровое изображение, а затем перенесите его в ту же папку, где находится `.cpp`-файл. Поскольку вы еще не сделали этого, можно выполнить эти операции позже, а затем удалить `.cpp`-файл из пакета и снова его добавить. Если в данной инструкции что-то неясно, найдите и прочитайте в справочных файлах C++Builder раздел “Adding Palette Bitmaps”. Теперь рассмотрим дополнения, которые необходимо ввести в объявление класса (листинг 30.16).

Листинг 30.16. Файл `JoyStick.h` с некоторыми модификациями

```
//---------------------------------------------------------------------
#ifndef JoyStickH
#define JoyStickH
//---------------------------------------------------------------------
#include <SysUtils.hpp>
#include <Controls.hpp>
#include <Classes.hpp>
#include <Forms.hpp>
#include <MMSYSTEM.h>
//---------------------------------------------------------------------
class PACKAGE TJoyStick : public TWinControl
{
private:
        int FDriverCount;
        int FXmin, FXmax, FYmin, FYmax, FPeriodmin, FPeriodmax;
        bool FConnected;
        int FJoyStickID;
        JOYCAPS FJoyCaps;
    System::AnsiString FName;
    System::AnsiString FMfrID;
    System::AnsiString FProdID;
        int FNumberButtons;
    TNotifyEvent FOnMove;
    TKeyPressEvent FOnButton1Down;
    TKeyPressEvent FOnButton2Down;
        int FX;
        int FY;
    void __fastcall SetConnected(bool value);
    void __fastcall mButtonUpdate(TMessage &msg);
    void __fastcall mMove(TMessage &msg);
protected:
public:
    __fastcall TJoyStick(TComponent* Owner);
    void __fastcall Connect(void);
    __fastcall ~TJoyStick(void);
__published:
    __property System::AnsiString Name = { read = FName, write = FName };
    __property bool Connected = { read = FConnected, write = SetConnected };
    __property int DriverCount = { read = FDriverCount };
    __property int Xmin = { read = FXmin };
    __property int Xmax = { read = FXmax };
    __property int Ymin = { read = FYmin };
    __property int Ymax = { read = FYmax };
    __property int Periodmin = { read = FPeriodmin };
    __property int Periodmax = { read = FPeriodmax };
    __property int JoyStickID = { read = FJoyStickID, write = FJoyStickID };
    __property int NumberButtons = { read = FNumberButtons };
    __property System::AnsiString Manufacturer = { read = FMfrID };
    __property System::AnsiString ProductID = { read = FProdID };
```

```
__property TNotifyEvent OnMove = { read = FOnMove, write = FOnMove };
__property TKeyPressEvent OnButton1Down =
          { read = FOnButton1Down, write = FOnButton1Down };
__property TKeyPressEvent OnButton2Down =
          { read = FOnButton2Down, write = FOnButton2Down };
__property int X = { read = FX, write = FX };
__property int Y = { read = FY, write = FY };
```

В первой части листинга 30.16 определяются переменные, предназначенные для внутреннего использования, и методы, которые мы хотим разделять с пользователем. Обратите внимание на то, что многие переменные определены с начальной буквой `F`. Мы также должны следовать этому соглашению, поскольку это облегчает понимание программы. Вероятно, вы заметили, что мы добавили три события. На этом этапе событие можно рассматривать просто как еще одну переменную.

Обратите также внимание, что в конце раздела `private` мы определили три метода, к которым доступ со стороны пользователей крайне нежелателен. В разделе `public` мы предоставили доступ пользователям нашего компонента только к процедурам инициализации и соединения.

Теперь поговорим о более интересных вещах. Мы должны определить свойства, к которым будет иметь доступ любой пользователь. Они помещены в раздел `__published` заголовочного файла `JoyStick.h` (см. листинг 30.16).

Интересно здесь то, что определенные нами события обрабатываются подобно простым свойствам данных. После ключевого слова `__property` мы вводим тип данных свойства, имя, которое будет видеть пользователь, знак равенства и заключенный в круглые скобки оператор `read`/`write`. Сюда можно было бы также добавить уточнение `default`/`stored`. При создании этой программы мы позаботились о том, чтобы иметь системную поддержку при чтении и записи большинства переменных. Это самый простой путь. В некоторых случаях, когда пользователь запрашивает значение свойства или пытается установить его, возможно, стоит выполнять некоторую обработку, т.е. написать простой метод и ввести имя этого метода в оператор `read`/`write`, как это сделано в свойстве `Connected` (для записи, вместо метода `Fconnected`, используется `SetConnected`). Свойства `X` и `Y` в листинге 30.16 возвращаются пользователю в единицах джойстика, а не экранных пикселях, как это делается обычно. Это — пример того, как метод, написанный вручную, имеет больше смысла, чем результат работы системных средств. В данном случае нам нужно ввести логику для преобразования единиц джойстика в пиксели.

```
return ((float)JoyStick1->X / JoyStick1->Xmax)* Width;
```

Кроме того, следует отметить, что в листинге 30.16 большинство свойств `JOYSTICK` определено только для чтения. (Если для свойства не определен атрибут `write =`, значит, это свойство предназначено только для чтения.)

Наконец, необходимо определить (после публикуемых свойств) карту сообщений (своего рода таблицу, реализующую механизм перенаправления системных сообщений и команд соответствующим объектам приложения), показанную в листинге 30.17.

Листинг 30.17. Карта сообщений для реакции на события (сообщения)

```
BEGIN_MESSAGE_MAP
    MESSAGE_HANDLER(MM_JOY1BUTTONDOWN, TMessage, mButtonUpdate);
    MESSAGE_HANDLER(MM_JOY1BUTTONUP, TMessage, mButtonUpdate);
    MESSAGE_HANDLER(MM_JOY2BUTTONDOWN, TMessage, mButtonUpdate);
    MESSAGE_HANDLER(MM_JOY2BUTTONUP, TMessage, mButtonUpdate);
```

```
    MESSAGE_HANDLER(MM_JOY1MOVE, TMessage, mMove);
    MESSAGE_HANDLER(MM_JOY2MOVE, TMessage, mMove);
END_MESSAGE_MAP(TWinControl)
};
#endif
```

Здесь нет никакой магии: мы просто создаем список сообщений Windows, которые собираемся обрабатывать, и указываем, что нужно делать по их получении.

Теперь пора рассмотреть код реализации. Прежде всего нужно обеспечить подключение к джойстику. Эта возможность реализуется с помощью кода, представленного в листинге 30.18.

Листинг 30.18. Метод `TJoyStick::Connect`, который инициализирует переменные и выполняет подключение к мультимедийному интерфейсу Windows (Windows Multimedia API)

```
void __fastcall TJoyStick::Connect(void)
{
    FConnected = false;
    MMRESULT     result;
    JOYINFOEX    joyinfo;
    FDriverCount = joyGetNumDevs();
    // Проверяем драйверы.
    if (FDriverCount != 0)
    {
        if (FJoyStickID <= JOYSTICKID1)
        {
            result = joyGetPosEx(JOYSTICKID1, &joyinfo);
            if (result == JOYERR_NOERROR)
            {
                FConnected = true;
                FJoyStickID = JOYSTICKID1;
            }
        }
        else
        {
            result = joyGetPosEx(JOYSTICKID2, &joyinfo);
            if (result == JOYERR_NOERROR)
            {
                FConnected = true;
                FJoyStickID = JOYSTICKID2;
            }
        }
        if (result == MMSYSERR_INVALPARAM)
        {
            Application->MessageBox(
                    "An error occured while connecting to joystick",
                    "An invalid parameter was passed.", MB_OK);
          }
          else if (result == MMSYSERR_NODRIVER)
          {
              Application->MessageBox(
                                "The joystick driver is not present.",
```

```
                              "Error", MB_OK);
          }
          else if (result == MMSYSERR_BADDEVICEID)
          {
               Application->MessageBox(
                     "The specified joystick identifier is invalid.",
                     "Error", MB_OK);
          }
          else if (result == JOYERR_UNPLUGGED)
          {
               Application->MessageBox(
                     "The specified joystick is not connected to the ↳ system.",
                           "Error", MB_OK);
          }
     }
     joyGetDevCaps(FJoyStickID, &FJoyCaps, sizeof(JOYCAPS));
     FMfrID = IntToStr(FJoyCaps.wMid);
     FProdID = IntToStr(FJoyCaps.wPid);
     FName = String(FJoyCaps.szPname);
     FXmin = FJoyCaps.wXmin;
     FXmax = FJoyCaps.wXmax;
     FX = (FXmin + FXmax) / 2;
     FYmin = FJoyCaps.wYmin;
     FYmax = FJoyCaps.wYmax;
     FY = (FYmin + FYmax) / 2;
     FNumberButtons = FJoyCaps.wNumButtons;
     FPeriodmin = FJoyCaps.wPeriodMin;
     FPeriodmax = FJoyCaps.wPeriodMax;
     if (Connected)
     {
          joySetCapture(Handle, FJoyStickID, 2*FJoyCaps.wPeriodMin, false);
     }
}
```

Метод `joyGetNumDevs()` определяет, сколько джойстиков установлено в системе. Если их нет вообще, то и проблем никаких нет. Предположив, что в системе все-таки существует какой-нибудь джойстик, мы постараемся получить от него текущую информацию, используя код листинга 30.19.

Листинг 30.19. Тестирование джойстиков в системе

```
if (FJoyStickID <= JOYSTICKID1)
{
    result = joyGetPosEx(JOYSTICKID1, &joyinfo);
    if (result == JOYERR_NOERROR)
    {
        FConnected = true;
        FJoyStickID = JOYSTICKID1;
    }
}
```

Если первая попытка окажется неудачной, попытаем "счастья" для JOYSTICKID2. Наконец, протестируем результат на предмет возвращенных ошибок. При обнаружении ошибки проинформируем об этом пользователя (листинг 30.20).

Листинг 30.20. Уведомление пользователя об ошибке при неудачном соединении

```
if (result == MMSYSERR_INVALPARAM)
{
    Application->MessageBox(
                    "An error occured while connecting to joystick",
                    "An invalid parameter was passed.", MB_OK);
}
else if (result == MMSYSERR_NODRIVER)
{
     Application->MessageBox(
            "The joystick driver is not present.", "Error", MB_OK);
}
```

Затем попытаемся установить переменные (свойства пользователя) и перехватить сообщения, генерируемые джойстиком, как показано в листинге 30.21.

Листинг 30.21. Инициализация переменных

```
joyGetDevCaps(FJoyStickID, &FJoyCaps, sizeof(JOYCAPS));
FMfrID = IntToStr(FJoyCaps.wMid);
FProdID = IntToStr(FJoyCaps.wPid);
FName = String(FJoyCaps.szPname);
FXmin = FJoyCaps.wXmin;
FXmax = FJoyCaps.wXmax;
FX = (FXmin + FXmax) / 2;
FYmin = FJoyCaps.wYmin;
FYmax = FJoyCaps.wYmax;
FY = (FYmin + FYmax) / 2;
FNumberButtons = FJoyCaps.wNumButtons;
FPeriodmin = FJoyCaps.wPeriodMin;
FPeriodmax = FJoyCaps.wPeriodMax;
if (Connected)
{
    joySetCapture(Handle, FJoyStickID, 2*FJoyCaps.wPeriodMin, false);
}
```

Процедура инициализации невелика по размеру, но она показана в листинге 30.22 для демонстрации метода, который вам стоит использовать.

Листинг 30.22. Пример защиты условно-бесплатного программного кода

```
__fastcall TJoyStick::TJoyStick(TComponent*Owner)
        :TWinControl(Owner)
{
    TDateTime *Then = new TDateTime(2000, 12, 31);
    if (Now() > *Then)
    {
```

```
        Application->MessageBox("Your trial period has expired.",
                                          "Contact DrDigits", MB_OK);
          delete Then;
          return;
     }
     Parent = (TWinControl *) Owner;
     Connect();
     delete Then;
}
```

Часто спрашивают, как защитить свои программные продукты от кражи. Если вы разработали набор компонентов для продажи, то у вас есть возможность вставить в каждый компонент проверку даты, показанную в листинге 30.22. Распространяйте свои компоненты в качестве пробных версий в виде доступных для загрузки `.H`-, `.LIB`-, `.BPL`- и `.BPI`-файлов, не предоставляя при этом исходного кода. Заложенное вами устройство охраны должно ликвидировать ваш компонент после 31 декабря 2000 года (эта дата использована в листинге 30.22) или в любой другой день, заданный вами продемонстрированным выше способом.

При нажатии на кнопку джойстика необходимо уведомить об этом пользователя. Связь между нажатой кнопкой и предпринимаемым нами действием определяется в карте сообщений (см. листинг 30.17). Для этого используется код, приведенный в листинга 30.23.

Листинг 30.23. Запрос информации от джойстика при нажатии его кнопки

```
void __fastcall TJoyStick::mButtonUpdate(TMessage &msg)
{
    char c1 = '1';
    char c2 = '2';
    JOYINFO joyinfo;
    joyGetPos(FJoyStickID, &joyinfo);
    FX = joyinfo.wXpos;
    FY = joyinfo.wYpos;
    if (joyinfo.wButtons & JOY_BUTTON1)
        if (OnButton1Down)
            OnButton1Down(this, c1);
    if (joyinfo.wButtons & JOY_BUTTON2)
        if (OnButton2Down)
            OnButton1Down(this, c2);
}
```

Обратите внимание, что перед уведомлением пользователя мы должны запросить данные о позиции джойстика. Ключевое значение, которое мы передадим пользователю, будет равно 1 или 2 (в зависимости от того, какая была нажата кнопка). Помимо тестирования нажатой кнопки, мы также должны убедиться, что пользователь заполнил информацию для события `OnButtonXDown`. Если анализируемое значение равно `NULL`, мы не будем генерировать это событие.

Практически такая же логика требуется для метода джойстика `OnMove`, как показано в следующем фрагменте программы.

```
void __fastcall TJoyStick::mMove(TMessage &msg)
{
    JOYINFO joyinfo;
```

```
    joyGetPos(FJoyStickID, &joyinfo);
    FX = joyinfo.wXpos;
    FY = joyinfo.wYpos;
    if (OnMove)
        OnMove(this);
}
```

Наконец, рассмотрим процедуру деструктора. Здесь нам нужно проверить подключение к джойстику и освободить соединение до разрушения самого объекта. В автоматически генерируемом определении класса не предусмотрено стандартного деструктора. Теоретически мы могли бы рассчитывать на деструктор класса-предка. Но, к сожалению, этот класс "ничего не знает" ни о каких джойстиках, поэтому он не сможет "убрать за нами" достаточно хорошо.

```
__fastcall TJoyStick::~TJoyStick(void)
{
    if (Connected)joyReleaseCapture(JoyStickID);
}
```

Нам понадобится каким-то образом протестировать элемент управления. Содержимое листинга 30.24 взято из файла `Unit1.cpp`, принадлежащего тестовой программе `TestJoy.bpr`. Обратите особое внимание на то, как используются значения `X` и `Y`. Значения `Xmax` и `Ymax` служат для преобразования `JoyStick`-единиц в пиксели. Чтобы понять работу программы, загрузите тестовый проект с компакт-диска, прилагаемого к этой книге, скомпилируйте и выполните тестовую программу.

На заметку

Чтобы скомпилировать и выполнить тестовую программу `TestJoy`, необходимо сначала инсталлировать пакет `TJoyStick`, чтобы он был доступен для C++Builder. Для этого запустите C++Builder, выберите из главного меню C++Builder команду File⇨Open (Файл⇨Открыть) и найдите пакет `TJoyStick.bpk` (если вы еще не создали собственный пакет, то наш находится на прилагаемом к книге компакт-диске, в папке `TJoyStick`). Откройте этот пакет, в диалоговом окне Package щелкните на кнопке Compile (Компилировать). После компиляции пакета щелкните на кнопке Install (Инсталлировать), и пакет (готовый к употреблению) будет помещен в палитру компонентов.

Листинг 30.24. Тестовая программа `TestJoy.bpr`

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

#include "Unit1.h"
//---------------------------------------------------------------
#pragma package(smart_init)
#pragma link "TJoyStick"
#pragma resource "*.dfm"
TForm1 *Form1;
//---------------------------------------------------------------
__fastcall TForm1::TForm1(TComponent* Owner)
          :TForm(Owner)
{
     JoyStick1->Connected = true;
}
```

```
//---------------------------------------------------------------
void __fastcall TForm1::JoyStick1Move(TObject *Sender)
{
     int x, y;
     x = ((float) JoyStick1->X / JoyStick1->Xmax) * Width;
     y = ((float) JoyStick1->Y / JoyStick1->Ymax) * Height;
     Canvas->TextOut(x, y, "+");
}
//---------------------------------------------------------------
void __fastcall TForm1::JoyStick1Button1Down(TObject *Sender, char &Key)
{
     int x, y;
     x = ((float) JoyStick1->X / JoyStick1->Xmax) * Width;
     y = ((float) JoyStick1->Y / JoyStick1->Ymax) * Height;
     Canvas->TextOut(x, y, "1");
}
//---------------------------------------------------------------
void __fastcall TForm1::JoyStick1Button2Down(TObject *Sender, char &Key)
{
     int x, y;
     x = ((float) JoyStick1->X / JoyStick1->Xmax) * Width;
     y = ((float) JoyStick1->Y / JoyStick1->Ymax) * Height;
     Canvas->TextOut(x, y, "2");
}
//---------------------------------------------------------------
```

Поскольку значения свойств `X` и `Y` возвращаются в `JoyStick`-единицах, необходимо преобразовать их в процентные составляющие от максимального значения в `JoyStick`-единицах, а затем использовать полученный результат для определения экранных значений `X` и `Y`. По-видимому, это следует сделать в методах `GetX()`/ `GetY()` — успеха вам!

Создание приложения с функциями утилиты системного мониторинга

Часто необходимо встроить в приложение средство системы мониторинга производительности, которое работает подобно приложению System Monitor, поставляемому с Windows. Например, если вы создаете Internet-приложение, вам обязательно захочется получить информацию об использовании центрального процессора, скорости модемного соединения и количестве переданных и полученных байт. В этом разделе мы покажем, как получить такие данные.

Немного о системных ресурсах Windows

В поставку Win9x входит приложение System Monitor (`sysmon.exe`), которое предоставляет средства контроля системных ресурсов и производительности.

Системные ресурсы существуют в виде коллекции объектов. Все объекты, которые в данный момент активны в системе, зафиксированы в реестре Windows (Windows Registry) под следующей ключевой записью.

```
HKEY_LOCAL_MACHINE\System\CurrentControlSet\Control\PerfStats\Enum
```

Каждый ключ реестра, который находится под этой записью, называется объектом. Эти объекты практически одинаковы во всех системах, например KERNEL, Dial-Up Adapter, VFAT и пр.

На заметку

Приложение System Monitor не инсталлируется по умолчанию в среде Win9x. Его можно инсталлировать, выполнив следующие действия.

1. Выберите команду Start⇨Settings⇨Control Panel (Пуск⇨Настройка⇨Панель управления).
2. Дважды щелкните на пиктограмме Add/Remove Programs (Установка и удаление программ).
3. Выберите вкладку Windows Setup (Установка Windows).
4. Выделите компонент System Tools (Служебные), а затем щелкните на кнопке Details (Состав).
5. Установите флажок опции System Monitor (Системный монитор) и щелкните на кнопке OK.

Ключ каждого объекта имеет строковое значение `Name`, которое идентифицирует обычно используемое имя этого объекта. Например, объект `VCACHE` имеет строковое значение имени `Disk Cache`, которое представляет его настоящее полное имя.

Под каждым ключом объекта находится некоторое количество подключей. Например, ключ Dial-Up Adapter имеет более дюжины подключей, которые называются *счетчиками*. Эти счетчики характеризуются тремя значениями реестра.

- Строковое значение `Name`, которое содержит полное имя счетчика.
- Строковое значение `Description`, которое содержит описательную информацию о счетчике.
- Строковое значение `Differentiate`, которое может быть равно либо `True`, либо `False`.

Значение `Differentiate` устанавливается равным `true` для обозначения того, что счетчик будет изменять свои показания с течением времени. Следовательно, чтобы получить значение счетчика, нужно следовать следующему правилу.

```
Counter_Value = (Counter_Value2-Counter_Value1) / (Time_Taken2-Time_Taken1)
```

Чтобы получить данные о производительности, приложение System Monitor прочитает их из следующего ключа системного реестра.

```
HKEY_DYN_DATA\PerfStats\
```

Чтобы прочитать данные о производительности счетчика, необходимо выполнить следующие действия.

1. Разрешаем сбор данных. Чтобы прочитать значение счетчика размером в 4 байта из ключевой записи реестра `HKEY_DYN_DATA\PerfStats\StartStat`, обращаемся к ключам вида *`Object\Counter`*. Например, чтобы разрешить сбор данных со счетчика `BytesRecvd` в объекте `Dial-Up Adapter`, необходимо прочитать двоичное значение `Dial-Up Adapter\BytesRecvd` из ключа реестра `KEY_DYN_DATA\PerfStats\StartStat`. При выполнении следующего фрагмента программы разрешается сбор данных с этого счетчика.

   ```
   Byte Dummy [4];
   TRegistry *Reg = new TRegistry;
   ```

```
Reg->RootKey = HKEY_DYN_DATA;
if (Reg->OpenKey("\\PerfStats\\StartStat", False))
{
     Reg->ReadBinaryData("Dial-Up Adapter\\BytesRecvd", Dummy, 4);
}
Reg->CloseKey();
```

2. Считываем данные о производительности. После инициализации коллекции данных пара *Object\Counter* используется для получения данных о производительности из ключа реестра KEY_DYN_DATA\PerfStats\StartData. При выполнении следующего фрагмента программы данные о производительности считываются для ранее инициализированной пары *Object\Counter*.

```
Byte Buffer [4];
TRegistry *Reg = new TRegistry;
Reg->RootKey = HKEY_DYN_DATA;
if (Reg->OpenKey("\\PerfStats\\StartData", False))
{
   Reg->ReadBinaryData("Dial-Up Adapter\\BytesRecvd", Buffer, 4);
}
Reg->CloseKey();
```

3. Запрещаем сбор данных. После считывания данных необходимо запретить сбор данных посредством чтения значения реестра, размещенного в ключе KEY_DYN_DATA\PerfStats\StopStat для пары *Object\Counter*. При выполнении следующего фрагмента программы сбор данных о производительности для пары *Object\Counter* прекращается.

```
Byte Dummy [4];
TRegistry *Reg = new TRegistry;
Reg->RootKey = HKEY_DYN_DATA;
if (Reg->OpenKey("\\PerfStats\\StopStat", False))
{
     Reg->ReadBinaryData("Dial-Up Adapter\\BytesRecvd", Dummy, 4);
}
Reg->CloseKey ();
```

Обратите внимание, что для получения корректного результата нужно повторить описанные выше действия, если значение Differentiation счетчика равно true. Каждый раз, перед чтением данных (согласно пункту 2), необходимо уведомить систему о своем намерении, как описано в пункте 1. Завершив чтение, проинформируйте систему об окончании чтения данных, как показано в пункте 3. Такие рекомендации дает компания Microsoft относительно чтения данных из системного реестра, и вы увидели, как эти рекомендации были выполнены.

Решение

Чтобы реализовать алгоритм, описанный в предыдущем разделе, мы создадим новый проект, который будет отображать использование центрального процессора (CPU), количество байт, переданных и принятых в секунду, и скорость соединения.

Начнем с создания нового приложения, добавим необходимые компоненты и установим их свойства, как показано в табл. 30.7. Напомню, что компонент TTimer находится во вкладке System, а компонент TCGauge — во вкладке Samples палитры компонентов.

Таблица 30.7. Компоненты для мониторинга системы и их свойства

Компонент	Свойство	Значения
TForm	Name	Form1
	Caption	System Monitor
TMainMenu	Name	MainMenu1
	MenuItem	&File
	MenuItem	&Exit
	MenuItem	&View
	MenuItem	Show CPU Usage Bar
	MenuItem	Show Received Bytes Bar
	MenuItem	Show Transmitted Bytes Bar
	MenuItem	Show Connection Speed Bar
TTimer	Name	Timer1
	Interval	1000 (1 секунда)
TPanel	Name	Panel1
	Align	alClient
TPanel	Name	CPUBar
	Align	alTop
TPanel	Name	RecvBar
	Align	alTop
TPanel	Name	XmitBar
	Align	alTop
TPanel	Name	ConnectSpeedBar
	Align	alTop
TCGauge (над панелью CPUBar)	Name	CPUGauge
	Min	0
	Max	100
TCGauge (над панелью RecvBar)	Name	RecvGauge
	Min	0
	Max	100
TCGauge (над панелью XmitBar)	Name	XmitGauge
	Min	0
	Max	100
TCGauge (над панелью ConnectSpeedBar)	Name	ConnectSpeedGauge
	Min	0

Окончание табл. 30.7

Компонент	Свойство	Значения
	Max	100
TLabel (над панелью CPUBar)	Name	Label1
	Left	13
	Top	7
TLabel (над панелью RecvBar)	Name	Label2
	Left	13
	Top	7
TLabel (над панелью XmitBar)	Name	Label3
	Left	13
	Top	7
TLabel (над панелью ConnectSpeedBar)	Name	Label4
	Left	13
	Top	7
TLabel (над панелью CPUBar)	Name	Label5
	Left	364
	Top	7
TLabel (над панелью RecvBar)	Name	Label6
	Left	364
	Top	7
TLabel (над панелью XmitBar)	Name	Label7
	Left	364
	Top	7
TLabel (над панелью ConnectSpeedBar)	Name	Label8
	Left	364
	Top	7
TLabel (над панелью RecvBar)	Name	Label9
	Left	13
	Top	40
TLabel (над панелью XmitBar)	Name	Label10
	Left	13
	Top	40
TLabel (над панелью ConnectSpeedBar)	Name	Label11
	Left	13
	Top	40

Дважды щелкните на таймере и введите код, содержащийся в листинге 30.25.

Листинг 30.25. Обработчик события `OnTimer` для элемента `Timer1`

```
void __fastcall TForm1::Timer1Timer(TObject *Sender)
{
    long CurBytesRec, CurBytesXmt;
    CPUGauge->Progress = ReadDataFromReg(CPUUsage);
    CurBytesRec = ReadDataFromReg(BytesRec);
    RecvGauge->Progress = CurBytesRec-LastBytesRec;
    Label9->Caption = FloatToStr(RecvGauge->Progress);
    LastBytesRec = CurBytesRec;
    CurBytesXmt = ReadDataFromReg(BytesXmt);
    XmitGauge->Progress = CurBytesXmt-LastBytesXmt;
    Label10->Caption = FloatToStr(XmitGauge->Progress);
    LastBytesXmt = CurBytesXmt;
    ConnectspeedGauge->Progress = ReadDataFromReg(ConnectSpeed);
    Label11->Caption = FloatToStr(ConnectspeedGauge->Progress);
}
```

Нетрудно заметить, что, в отличие от данных, касающихся центрального процессора и скорости соединения, которые читаются напрямую, данные о переданных и принятых байтах нужно получать путем вычитания из значений, считанных накануне. Как пояснялось выше, все дело в том, что строковые значения `Differentiate` этих ключей устанавливаются равными `true`. Таким образом, мы должны использовать следующую формулу.

```
Counter_Value = (Counter_Value2-Counter_Value1) / (Time_Taken2-Time_Taken1);
```

Используемый в этой формуле промежуток времени между считываниями данных равен 1, поскольку интервал для элемента `Timer1` устанавливается равным 1000 (1 секунда). Следовательно, эту формулу можно переписать следующим образом.

```
Counter_Value = Counter_Value2-Counter_Value1;
```

Основной в этом событии и программе в целом является функция `ReadDataFromReg()`. Она получает требуемые данные из реестра путем инициализации начала сбора данных, чтения данных о производительности и, наконец, остановки сбора данных. Она попросту реализует все, о чем шла речь выше, и это представлено в листинге 30.26.

Листинг 30.26. Функция `ReadDataFromReg()`

```
long __fastcall TForm1::ReadDataFromReg(const char* ObjectName)
{
    Byte Buffer [4], Dummy [4];
    long *Bytes;
    if (Reg1->OpenKey(StartKey, False))
    {
        Reg1->ReadBinaryData(ObjectName, Dummy, 4);
    }
    Reg1->CloseKey();
    if (Reg1->OpenKey(DataKey, False))
    {
        Reg1->ReadBinaryData(ObjectName, Buffer, 4);
        Bytes = (long*)&Buffer;
```

```
    }
    Reg1->CloseKey();
    if (Reg1->OpenKey(StopKey, False))
    {
        Reg1->ReadBinaryData(ObjectName, Dummy, 4);
    }
    Reg1->CloseKey();
    return *Bytes;
}
```

Остальной код в файле `Unit1.cpp` (листинг 30.27) представляет собой инициализацию переменных, значения которых определяют внешний вид панелей в форме.

Листинг 30.27. Остальной код файла `Unit1.cpp`

```
#include <vcl.h>
#pragma hdrstop
#include "Unit1.h"
//---------------------------------------------------------------
#pragma package(smart_init)
#pragma link "CGAUGES"
#pragma resource "*.dfm"
TForm1 *Form1;
const AnsiString
StartKey = "\\PerfStats\\StartStat",
DataKey = "\\PerfStats\\StatData",
StopKey = "\\PerfStats\\StopStat",
CPUUsage = "KERNEL\\CPUUsage",
BytesRec = "Dial-Up Adapter\\BytesRecvd",
BytesXmt = "Dial-Up Adapter\\BytesXmit",
ConnectSpeed = "Dial-Up Adapter\\ConnectSpeed",
PerfStatsDial =
     "\\System\\CurrentControlSet\\control\\PerfStats\\Enum\\Dial-Up Adapter",
PerfStatsKernel =
 "\\System\\CurrentControlSet\\control\\PerfStats\\Enum\\KERNEL";
long LastBytesXmt,LastBytesRec;
TRegistry *Reg1,*Reg2;
//---------------------------------------------------------------
__fastcall TForm1::TForm1(TComponent* Owner)
         : TForm(Owner)
{
     String name;
     String temp1,temp2;
     LastBytesRec = 0;
     LastBytesXmt = 0;

     Reg1 = new TRegistry;
     Reg1->RootKey = HKEY_DYN_DATA;

     Reg2 = new TRegistry;
     Reg2->RootKey = HKEY_LOCAL_MACHINE;
```

```
     temp1 = PerfStatsKernel;
     temp2 = temp1 + "\\CPUUsage";
     if (Reg2->OpenKey(temp1, False))
          name = Reg2->ReadString("Name");
     Reg2->CloseKey();

     if (Reg2->OpenKey(temp2, False))
          Label1->Caption = name + ": " + Reg2->ReadString("Name");
     Reg2->CloseKey();

     temp1 = PerfStatsDial;
     if (Reg2->OpenKey(temp1, False))
          name = Reg2->ReadString("Name");
     Reg2->CloseKey();

     temp2 = temp1 + "\\BytesRecvd";
     if (Reg2->OpenKey(temp2, False))
          Label2->Caption = name + ": " + Reg2->ReadString("Name");
     Reg2->CloseKey();

     temp2 = temp1 + "\\BytesXmit";
     if (Reg2->OpenKey(temp2, False))
          Label3->Caption = name + ": " + Reg2->ReadString("Name");
     Reg2->CloseKey();

     temp2 = temp1 + "\\ConnectSpeed";
     if (Reg2->OpenKey(temp2, False))
          Label4->Caption = name + ": " + Reg2->ReadString("Name");
     Reg2->CloseKey();
}
//---------------------------------------------------------------
void __fastcall TForm1::FormDestroy(TObject *Sender)
{
     delete Reg1;
     delete Reg2;
}
//---------------------------------------------------------------
void __fastcall TForm1::ShowCPUUsageBarClick(TObject *Sender)
{
     ShowCPUUsageBar->Checked = !ShowCPUUsageBar->Checked;
     if(ShowCPUUsageBar->Checked)
          CPUBar->Visible = true;
     else
          CPUBar->Visible = false;
}
//---------------------------------------------------------------
void __fastcall TForm1::ShowReceivedBytesBarClick(TObject *Sender)
{
     ShowReceivedBytesBar->Checked = !ShowReceivedBytesBar->Checked;
     if(ShowReceivedBytesBar->Checked)
          RecvBar->Visible = true;
```

```
     else
          RecvBar->Visible = false;
}
//---------------------------------------------------------------
void __fastcall TForm1::ShowTransmittedBytesBarClick(TObject *Sender)
{
     ShowTransmittedBytesBar->Checked = !ShowTransmittedBytesBar->Checked;
     if(ShowTransmittedBytesBar->Checked)
          XmitBar->Visible = true;
     else
          XmitBar->Visible = false;
}
//---------------------------------------------------------------
void __fastcall TForm1::ShowConnectionSpeedBarClick(TObject *Sender)
{
     ShowConnectionSpeedBar->Checked = !ShowConnectionSpeedBar->Checked;
     if(ShowConnectionSpeedBar->Checked)
          ConnectspeedBar->Visible = true;
     else
          ConnectspeedBar->Visible = false;
}
//---------------------------------------------------------------
void __fastcall TForm1::ExitClick(TObject *Sender)
{
     Application->Terminate();
}
```

Полный текст программы `SysMonitor` находится на компакт-диске, прилагаемом к этой книге. На рис. 30.10 программа `SysMonitor` показана в действии.

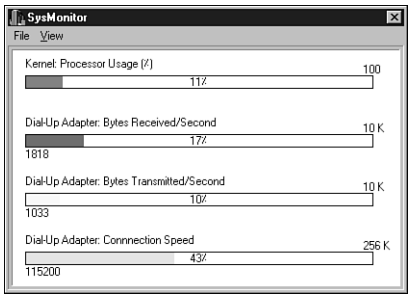


Рис. 30.10. Программа `SysMonitor`

Резюме по системному мониторингу

Из реестра можно почерпнуть большой объем информации о производительности системы. Более того, используя функцию `RegConnectRegistry()`, можно прочитать данные о производительности даже удаленного компьютера, поскольку эта функция позволяет получать доступ (как для чтения, так и для записи) к удаленному системному реестру.

Программа SysMonitor — чрезвычайно проста и может служить ориентиром для дальнейшей работы в этом направлении. Чтобы получить представление о том, какой объем информации можно прочитать из системного реестра, познакомьтесь поближе с программой SysMon.exe, которая входит в поставку Windows.

Исследование приложения Soundex

Программа Soundex была разработана в начале прошлого века, когда резко возрос приток эмигрантов в Соединенные Штаты. Эти эмигранты носили странные для американцев имена: вместо привычного для уха Джона Смита то и дело слышалось Сидлоски Джульф (Sydloski Juilf). Государственные служащие никогда и не предполагали о таких возможностях произношения, и даже корректор орфографии Microsoft Word "спотыкался" на таких именах. Для упрощения процесса ввода подобных имен и фамилий была создана программа Soundex. Вот как она работает.

- 1 = B, P, F, V
- 2 = C, S, K, G, J, Q, X, Z
- 3 = D, T
- 4 = L
- 5 = M, N
- 6 = R
- Буквы A, E, I, O, U, W, Y, H игнорируются.

Итак, оставляем начальную букву фамилии.

- Удаляем удвоенные буквы.
- Переводим следующие буквы с помощью приведенного выше списка, присоединяя, таким образом, к первой букве три цифры.
- Если в имени есть такой префикс, как ван (Van), вон (Von), де (De), ди (Di), ле (Le) или ла (La), то его можно оставить или исключить (по вашему усмотрению). Буквосочетания Мк (Mc) и Мак (Mac) не считаются префиксами и вводятся без последующего пробела.

В результате Pedersen переводится в Soundex-код как P362.
P + (D = 3), (R = 6), (S = 2)
Такая же "судьба" постигнет и Peterson, Patterson, Peters и Padorski.

Решение

Рассмотрим небольшую программу, использующую класс Soundex. Программа находится в папке SoundLike компакт-диска, прилагаемого к этой книге. Обратите внимание, что, хотя здесь не рассматриваются аспекты, связанные с хранением информации в базе данных, файл базы данных soundex.mdb приводится в той же папке. Чтобы скомпилировать и запустить программу, необходимо перед открытием проекта Soundex.bpr установить в C++Builder псевдоним для этой базы данных. Для этого выполните следующие действия.

1. Из главного меню C++Builder выберите команду Database⇨Explore (База данных⇨Исследовать).
2. В диалоговом окне Database Explorer (Проводник базы данных), выберите команду Object⇨ODBC Administrator (Объект⇨Администратор ODBC).

3. Щелкните на кнопке Add (Добавить) во вкладке User DSN диалогового окна ODBC Data Source Administrator (Администратор источника данных ODBC).
4. Выберите из списка элемент Microsoft Access Driver (.mdb) и щелкните на кнопке Finish (Готово).
5. Введите в диалоговом окне ODBC Microsoft Access в качестве имени источника данных (в поле Data Source Name) слово `Soundex`.
6. Щелкните на кнопке Select (Выбрать) и перейдите в папку, где хранится файл `Soundex.mdb`. Уже было отмечено, что этот файл находится в папке `SoundsLike` на прилагаемом к книге компакт-диска, но прежде чем выбирать файл `Soundex.mdb`, вы должны скопировать его с компакт-диска на свой жесткий диск.
7. Щелкайте на кнопке OK во всех диалоговых окнах, пока не окажетесь в диалоговом окне Database Explorer.
8. Из меню диалогового окна Database Explorer выберите команду Object⇨New (Объект⇨Создать).
9. Из раскрывающегося списка выберите элемент Microsoft Access Driver (.mdb) и щелкните на кнопке OK.
10. В результате во вкладке Databases (Базы данных) диалогового окна Database Explorer будет создан новый псевдоним с именем наподобие `ODBC1` . Замените это имя именем `Soundex`.
11. Из меню диалогового окна Database Explorer выберите команду Object⇨Apply (Объект⇨Применить). Если вам предложат сохранить изменения, щелкните на кнопке OK.
12. При выделенном псевдониме `Soundex` во вкладке Databases диалогового окна Database Explorer прокрутите вниз окно Definition (Определение), чтобы найти элемент ODBC DSN и выберите из раскрывающегося списка элемент `Soundex`.
13. Из меню Database Explorer выберите команду Object⇨Apply. Если вам предложат сохранить изменения, щелкните на кнопке OK.
14. Закройте окно Database Explorer. Можете прямо сейчас загрузить, скомпилировать и выполнить проект `Soundex` из папки `SoundsLike` компакт-диска, прилагаемого к книге.

Реализация класса `Soundex` представлена в листинге 30.28. Ниже мы рассмотрим несколько методов, которые могут оказаться весьма полезными, например фоновое затенение в окне списка, показанное на рис. 30.11. Вы можете открыть проект `Soundex` в C++Builder и просматривать текст программы во время ее обсуждения. Листинг 30.28 содержит определение класса `Soundex`.

Рис. 30.11. Программа `Soundex` в действии

Листинг 30.28. Файл `Soundex.h`. Определение класса

```
#ifndef SoundexH
#define SoundexH
//---------------------------------
class Soundex :public TObject
{
__published:
private:
    System::AnsiString Code; // Здесь будет наша программа.
public:
    __fastcall Soundex(System::AnsiString Name);
    void __fastcall Set(System::AnsiString Name);
    System::AnsiString __fastcall Get(void);
};

//---------------------------------
__fastcall Soundex::Soundex(System::AnsiString Name)
{
    Set(Name); // Инициализация класса.
}

//---------------------------------
System::AnsiString __fastcall Soundex::Get(void)
{
    return Code; // Для получения кода.
}

//---------------------------------
void __fastcall Soundex::Set(System::AnsiString Name)
{
    char hold [10];
    Code = "";
     int index, len;
     char c;

     while (Name.SubString(1,1).UpperCase() ==
           Name.SubString(2,1).UpperCase()) // Удаление дубликатов
    {
        Name.Delete(1,1);
    }

    Code += Name.SubString(1,1).UpperCase();//Берем первый символ
    len =Name.Length();
    index = 2;   // Начинаем с просмотра второго символа.

    // Продолжаем до тех пор, пока не получим код или не
    // закончится имя.
    while ((Code.Length() < 4) && (len > 0) && (index <= len))
    {
        while (Name.SubString(index,1).UpperCase() ==
```

```
                    Name.SubString(index+1, 1).UpperCase())
        {
            Name.Delete(index,1);
        }
    }
    len = Name.Length();

    strcpy(hold, Name.SubString(index, 1).LowerCase().c_str());
    c = hold[0];
    switch (c)   // В зависимости от символа, наращиваем код
                 // или пропускаем.
    {
        case 'b': case 'p': case 'f': case 'v':
              Code += "1";
              index++;
              break;
         case 'c': case 's': case 'k': case 'g': case 'j':
                                       case 'q': case 'x': case 'z':
              Code += "2";
              index++;
              break;
         case 'd': case 't':
              Code += "3";
              index++;
              break;
         case 'l':
              Code += "4";
              index++;
              break;
         case 'm': case 'n':
              Code += "5";
              index++;
              break;
         case 'r':
              Code += "6";
              index++;
              break;
         default:
              index++;
              break;
     }
}
//-----------------------------------
#endif
```

Рассмотрим главную форму проекта. Строка редактирования в верхней части первого экрана используется для ввода нужного имени.

Мы следим за тем, какие клавиши нажимает пользователь, и перехватываем нажатие клавиши <Enter>, что позволяет вводить такие данные, как имена, в более естественном виде. Для этого мы используем двойной щелчок на этом элементе управления и событие `OnKeyDown`.

Код обработки нажатий клавиш показан в листинге 30.29.

Листинг 30.29. Действия, выполняемые при нажатии пользователем клавиши <Enter>

```
Soundex *Code;
int index;
if (Key == VK_RETURN)
{
    ListBox1->Clear();
    Code = new Soundex(Edit1->Text);
    DM->Query1->Active = false;
    DM->Query1->Params->Items[0]->AsString = Code->Get();
    DM->Query1->Open();
    DM->Query1->First();
    while (!DM->Query1->Eof)
    {
        index = ListBox1->Items->Add(
           DM->Query1->FieldByName("Lastname")->AsString + "," ++
              DM->Query1->FieldByName("Firstname")->AsString);
         ListBox1->Items->Objects[index] == (
              TObject *)DM->Query1->FieldByName("Key")->AsInteger;
         DM->Query1->Next();
     }
     delete Code;
}
```

В третьей строке мы проверяем факт нажатия клавиши <Enter>. Обычно это означает, что пользователь завершил ввод данных. Убедившись в нажатии именно клавиши <Enter>, очищаем окно списка, создаем новый объект `Soundex` и используем его для ввода параметров запроса. SQL-оператор в этом запросе выглядит следующим образом.

```
SELECT * FROM Customers C
WHERE :Code = C.Soundex ORDER BY C.Lastname ASC
```

За дополнительной информацией о запросах SQL обращайтесь к разделу “SQL” главы 14.

Открыв запрос, программа в цикле просматривает каждую запись, добавляя в окно списка имя заказчика, а в коллекцию объектов этого списка — ключевое значение заказчика. Это ключевое значение, или ключ, нам понадобится уже через мгновение.

Теперь взглянем на затенение в списке (см. рис. 30.11). Необходимость такого затенения может быть вызвана различными причинами. Обычно затеняются выбранные записи, чтобы выделить их на фоне других. Выделять данные также можно по причине их исключительности или просто для облегчения просмотра. Это делается с помощью метода `DrawItem()`, которым обладает окно списка. Для этого нужно сначала установить стилевое свойство окна списка. В данном случае ему присваивается значение `lbOwnerDrawFixed`, поскольку мы собираемся сгенерировать событие, но не планируем изменять размеры текста. Если бы мы задали значение `lbOwnerDrawVariable`, это позволило бы менять размеры каждой строки и помещать в окно списка растровые изображения. Процедура рисования чрезвычайно проста. Как и в случае использования строки редактирования, мы дважды щелкаем на нужном событии. Обратите внимание, что событие `OnClick` также задействовано.

```
void __fastcall TForm1::ListBox1DrawItem(TWinControl *Control,
                  int Index, TRect &Rect, TOwnerDrawState State)
```

```
{
    TCanvas *pCanvas = ((TListBox *)Control)->Canvas;
    if (Index &0x01) pCanvas->Brush->Color = clLime;
    else pCanvas->Brush->Color = clLtGray;
    pCanvas->FillRect(Rect);
    pCanvas->TextOut(Rect.Left + 2,Rect.Top,((
                    TListBox *)Control)->Items->Strings[Index]);
}
```

Оператор `if` проверяет на четность номер строки в списке. Для строк с нечетными номерами устанавливается определенный цвет кисти для канвы окна списка. Обратите внимание, что для получения доступа к адресу элемента типа `TCanvas` переданный методу адрес `Control` приводится к типу указателя на `TListBox`. После установки цвета кисти используется метод `FillRect()` для окраски фона, а текстовая строка записывается на канву с помощью метода `TextOut()`.

Когда пользователь решит для себя, какой элемент списка является правильным, он должен уведомить о своем решении программу, щелкнув на нужном элементе. Обработчик события `OnClick`, представленный в листинге 30.30, позволяет открыть для пользователя вторую вкладку этой программы.

Листинг 30.30. Изменение курсора базы данных и переход на другую вкладку

```
void __fastcall TForm1::ListBox1Click(TObject *Sender)
{
    for (int i = 0; i < ListBox1->Items->Count; i++)
    {
        if (ListBox1->Selected[i])
        {
            DM->Table1->SetKey();
            DM->Table1Key->AsInteger = (long)ListBox1->Items->Objects[i];
            if (DM->Table1->GotoKey())
            {
                PageControl1->ActivePage = TabSheet2;
                return;
            }
        }
    }
}
```

Когда пользователь щелкает на элементе списка, строка с этим элементом становится выделенной. В цикле можно легко опросить все элементы списка и найти выбранную пользователем строку. Индекс выбранной строки можно затем использовать для считывания ключа заказчика из коллекции объектов списка, а этот ключ, в свою очередь, используется для установки ключа таблицы `TTable`. Если функция `GoToKey` устанавливает курсор файла на желаемую запись, мы открывает вторую вкладку окна программы.

Во второй вкладке отображается запись одного заказчика, и для отображения сделанных им покупок используется элемент типа `DBGrid`. Процесс отображения облегчается благодаря использованию второго элемента `TTable`, который связан с первой таблицей связью типа “главная/подчиненная”. В действительности, когда мы установили ключ первой таблицы и выполнили функцию `GoToKey`, ко второй таблице автоматически был применен фильтр, чтобы в ней остались только записи, содержащие ключ заказчика.

Таблица Table1 определена в качестве главной для таблицы Table2. Ее свойства MasterSource и MasterField установлены для указания на источник данных Table1.

Элемент DBGrid можно использовать для ввода новых данных в таблицу purchases (Table2), используя клавишу <Tab> для перехода на новую строку.

Для ввода в базу данных информации о новых заказчиках можно использовать панель навигации (с кнопками переходов), расположенную в нижней части экрана. Именно на этом этапе (т.е. при вводе или изменении фамилии) для каждого нового заказчика вычисляется Soundex-код, который, однако, мы не считаем нужным отображать. Нет никакой необходимости в том, чтобы пользователь или служащий знал о его существовании (вносимые нами изменения — не для постороннего глаза). Чтобы облегчить процесс внесения изменений при выборе поля с фамилией заказчика, можно использовать редактор полей таблицы TTable. Этот редактор вызывается в результате щелчка правой кнопкой мыши на элементе TTable.

Выбрав поле с фамилией заказчика и нажав клавишу <F11>, мы активизируем редактор свойств для объекта TField. Затем можно дважды щелкнуть на событии OnChange этого поля, чтобы связать с ним необходимые действия. Это позволит "втайне от всех" вычислить Soundex-код и записать его значение в другое поле записи. В следующем фрагменте программы показана реализация этих действий.

```
void __fastcall TDM::Table1LastnameChange(TField *Sender)
{
    Soundex *a = new Soundex(Table1Lastname->AsString);
    Table1Soundex->AsString = a->Get();
    delete a;
}
```

Обратите внимание, что этот код размещается непосредственно в модуле данных, а не модуле формы.

Теперь подведем некоторые итоги. Программа Soundex не предназначена для коммерческих целей, а приведена здесь лишь для демонстрации используемых в ней методов. Коммерческая направленность предполагает гораздо больший объем работы по сравнению с реализованным здесь. Например, мы должны были бы добавить меню, справочные файлы и процедуры обработки прерываний. Однако мы надеемся, что несмотря на отмеченные недостатки, вы открыли для себя что-то новое и смогли убедиться в "красноречии" C++Builder.

Использование компонентов просмотра дерева

Средство просмотра дерева часто используется в таких программах, как Windows Explorer, различных почтовых утилитах и сотнях других приложений. Будучи встроенным в приложение, это средство просмотра обеспечивает весьма эффективное отображение структуры каталогов, почтовой информации, разделов энциклопедического словаря и различных элементов базы данных. Независимо от типа данных, они отображаются таким образом, что пользователю понятна иерархия данных и логические связи между элементами. Кроме того, средство просмотра дерева предоставляет пользователю возможность манипулировать данными в структурном формате, сворачивая и разворачивая узлы и ветви узлов.

Вполне возможно, что ваше приложение значительно бы выиграло, обладая таким средством просмотра элементов дерева. К счастью, C++Builder позволяет разработчикам легко включить его в приложение с помощью набора VCL-компонентов фирмы Borland TTree. В состав этого набора входят такие компоненты, как TTreeView, TTreeNodes и TTreeNode. В

этом разделе мы рассмотрим способы эффективного использования этих компонентов, в частности для управления объектами данных и записями. В качестве примера будет продемонстрировано простое "генеалогическое" приложение (программа Family Tree), в котором реализовано много идей и принципов, описанных в этом разделе (рис. 30.12). Исходный код программы Family Tree можно найти на прилагаемом к книге компакт-диске (в папке `Trees` раздела, отведенного для главы 30).

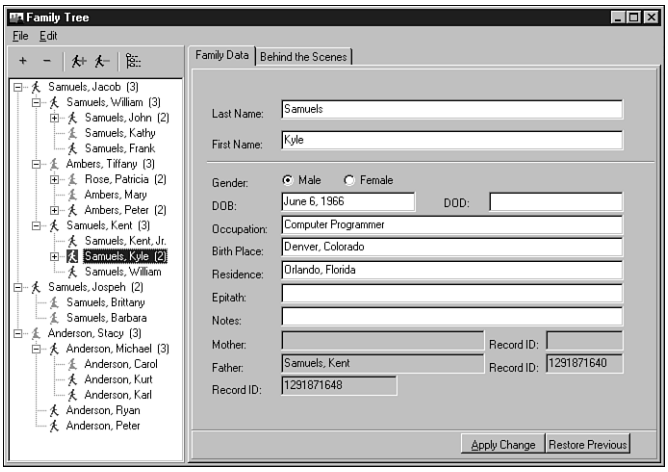


Рис. 30.12. Программа Family Tree, использующая средство просмотра дерева

Основы организации средства просмотра дерева

Прежде чем углубляться в более сложные темы, стоит хорошо разобраться в основах создания таких средств просмотра древовидных элементов. Полное описание визуального средства просмотра деревьев содержится в структуре `TTreeView`. Компонент `TTreeView` можно выбрать на панели инструментов VCL Win32 и опустить в форму при разработке приложения. Компонент `TTreeView` характеризуется свойством `Items`, которое определяется классом `TTreeNodes`. Свойство `Items` представляет все невизуальные аспекты средства просмотра дерева. Свойство `Items` содержит массив `Item`, представляющий все элементы данных (узлы), составляющие дерево. Каждый узел (или элемент) в массиве `Item` определяется классом `TTreeNode`. В каждом `TTree`-классе предусмотрен ряд ключевых методов и свойств, которые будут описаны ниже.

Добавление узлов

Прежде всего нужно научиться заполнять средство просмотра дерева данными. В классе `TTreeNodes` для добавления узлов предусмотрены следующие методы.

- `Add()`
- `AddChild()`
- `AddChildFirst()`
- `AddChildObject()`
- `AddChildObjectFirst()`
- `AddFirst()`
- `AddObject()`
- `AddObjectFirst()`

Метод AddChild() чрезвычайно полезен, поскольку устанавливает "потомственную" связь в средстве просмотра дерева. Если дерево пустое, метод AddChild() создает новый корневой узел в средстве просмотра дерева. Рассмотрим следующий фрагмент программы, в котором для добавления узлов к дереву используется метод AddChild().

```
AnsiString text = AnsiString("Person" +
                    AnsiString(TreeViewFamily->Items->Count));
TTreeNode* tempnode = TreeViewFamily->Items->AddChild(
                              TreeViewFamily->Selected, text);
tempnode->Selected = true; //Выбираем только что добавленный узел
```

Обратите внимание, что метод AddChild() принимает два параметра: указатель на узел дерева, определенный классом TTreeNodes, и идентификатор AnsiString. Первый параметр идентифицирует родительский узел, к которому будет добавлен дочерний. В рассматриваемом примере программы свойство TTreeViews->Selected используется для идентификации родителя (или ответвления), для которого будет добавлен узел. При первом проходе (когда средство просмотра дерева еще пустое) свойство Selected равно значению NULL, поскольку в дереве нет данных, которые можно было бы выбрать. К счастью, метод AddChild() построен таким образом, что в этом случае (т.е. когда он сталкивается с NULL-параметром) он создает новый корневой узел (в противном случае потребовалось бы вызывать метод Add()). Второй параметр задает текстовую надпись AnsiString для нового узла. Эта надпись будет отображена в самом средстве просмотра дерева. Метод AddChild() возвращает указатель на созданный им новый узел. Новый узел можно выделить внутри дерева, если установить свойству Selected значение true, как показано в предыдущем примере.

Недостатком использования метода AddChild() является возможность ввода только текстовой надписи и иерархической связи. Это ограничение касается и методов Add(), AddFirst() и AddFirstChild(). Часто в приложение (подобное рассматриваемому здесь генеалогическому дереву) хотелось бы поместить более расширенные данные (например, дату рождения, дату смерти, сведения о родителях, местожительство и род занятий). В состав стандартных библиотек VCL включены мощные средства, которые широко используются большинством программистов. Одно из них встроено в класс TTreeNodes. Используя методы AddObject(), AddChildObject(), AddObjectFirst() или AddChildObjectFirst(), разработчик может более эффективно управлять данными в программе. Объект, содержащийся в компоненте TTreeNode, является просто указателем на адрес реализованного элемента данных. Итак, рассмотрим объявление и параметры, принимаемые методом AddChildObject().

```
TTreeNode*__fastcall AddChildObject(TTreeNode*Node,
                          const System::AnsiStringS, void *Ptr);
```

За исключением параметра void* Ptr, первые два параметра метода совпадают с параметрами метода AddChild(). В качестве третьего параметра мы должны передать ссылку (указатель) на существующий объект. В результате будет подготовлен узел для объекта данных. В листинге 30.31 показано использование метода AddChildObject().

Листинг 30.31. Добавление узла дерева с данными

```
void __fastcall TFormFamilyTree::ActionNodeAddDataExecute(TObject *Sender)
{
     AnsiString text = AnsiString(
               "Person" + AnsiString(TreeViewFamily->Items->Count));
     ClearEntryPanel(); // Очищаем строку редактирования.
```

```
    // Создаем запись с данными для узла дерева.
    TPersonalInfo * PersonalInfo = new TPersonalInfo;
    PersonalInfo->ID = GetNewID(); // Получаем уникальный id.
    PersonalInfo->LastName = text;

    if (TreeViewFamily->Selected)
        if (TreeViewFamily->Selected->Data)
        {
            TPersonalInfo *ParentData;
            ParentData = (TPersonalInfo*)( TreeViewFamily->Selected->Data);
            PersonalInfo->LastName = ParentData->LastName;
            PersonalInfo->FirstName = text;

            if (ParentData->Gender) // Если родителем является
                       // мужчина, заполняем информацию об отце.
            {
                PersonalInfo->Father.Name = ParentData->Name;
                PersonalInfo->Father.ID = ParentData->ID;
            }
            else
            {
                PersonalInfo->Mother.Name = ParentData->Name;
                PersonalInfo->Mother.ID = ParentData->ID;
            }
        }
    TransferPersonalInfoToEditEntry(PersonalInfo);
    // Отображаем запись с данными в строке редактирования.

    // Вызываем процедуру для создания текстовой надписи для
    // нового узла дерева.
    PersonalInfo->Name = CreateLabel(EditFirstName->Text,EditLastName->Text);
    TTreeNode* tempnode = TreeViewFamily->Items->AddChildObject(
                  TreeViewFamily->Selected, PersonalInfo->Name,
                  PersonalInfo);
    if (PersonalInfo->Gender)
    {
        tempnode->ImageIndex = gPersonMale;
        tempnode->SelectedIndex = gPersonMaleSelected;
    }
    else
    {
        tempnode->ImageIndex = gPersonFemale;
        tempnode->SelectedIndex = gPersonFemaleSelected;
    }
    tempnode->Selected = true; // Выбираем только что добавленный узел.
}
```

Этот пример очень похож на приведенный выше пример использования метода `AddChild()`. Разница между ними в том, что в этом генеалогическом дереве используются дополнительные данные. После создания объекта записи в памяти его адрес с помощью метода `AddChildObject()`

можно передать соответствующему узлу дерева. Переменная указателя PersonalInfo определяется в соответствии с типом TPersonalInfo, который представляет собой нестандартную структуру, используемую для индивидуальных генеалогических данных (см. текст программы famtree.cpp, которая находится на компакт-диске, прилагаемом к этой книге). Переменная PersonalInfo инициализируется в начале обработчика, но не удаляется в конце, как можно было бы ожидать. Если бы она удалялась, адрес, сохраняемый новым узлом дерева, был бы неверным — по нему нельзя было бы найти нужных данных. Как будет показано ниже, запись PersonalInfo будет удаляться только в том случае, когда удаляется узел дерева или очищается все средство просмотра дерева.

Использование значков

C++Builder позволяет вставлять для узлов дерева графические пиктограммы (значки) и изменять их. В листинге 30.31 предусмотрена вставка такого значка после создания нового узла. Это реализуется с помощью свойств узлов дерева ImageIndex и SelectedIndex.

```
if (PersonalInfo->Gender)
{
    tempnode->ImageIndex = gPersonMale;
    tempnode->SelectedIndex = gPersonMaleSelected;
}
else
{
    tempnode->ImageIndex = gPersonFemale;
    tempnode->SelectedIndex = gPersonFemaleSelected;
}
```

Свойство ImageIndex идентифицирует значок, который должен отображаться, когда узел не выбран. Свойство SelectedIndex, с другой стороны, используется для идентификации значка, который необходимо отобразить, когда узел выбран. Сами значки хранятся в списке ImageList, который назначается дереву с помощью свойства Images. В нашем примере программы каждый значок представляется целочисленным идентификатором gPersonMale. Эти идентификаторы определяются как константы в заголовочном файле формы и иллюстрируются в тексте программы, находящейся на прилагаемом к книге компакт-диске (см. файл проекта famtree.bpr, расположенный в подпапке Trees папки, отведенной для главы 30). Если в качестве индекса используется значение –1, значок не отображается.

Узел дерева может поддерживать четыре различных типа свойства Image:

- ImageIndex
- SelectedIndex
- OverlayIndex
- StateIndex

Свойство OverlayIndex используется для представления прозрачной маски, которая перекрывает значок, отображаясь поверх выбранного в данный момент стандартного изображения значка. Чтобы корректно использовать свойство OverlayIndex, важно сначала преобразовать индексы изображений внутри списка значков для создания списка виртуальных масок. Это реализуется с помощью метода Overlay(), определенного в классе TCustomImageList, следующим образом.

```
void __fastcall TFormFamilyTree::FormActivate(TObject *Sender)
{
    ImageListTreeViewIcons->Overlay(gDeleted, 0);
}
```

В список виртуальных масок можно добавить до четырех индексов. Если свойство OverlayIndex содержит значение –1, маска удаляется.

Свойство StateIndex используется для отображения дополнительного значка для узла, который размещается слева от стандартного изображения. Свойство StateIndex предназначено для визуального представления состояния узла, в то время как свойства ImageIndex и SelectedIndex служат для визуального определения типа узла. Свойство ImageList, содержащее значки, идентифицируется свойством StateImages в средстве просмотра дерева. Список значков, используемый для свойства Images, можно использовать и для свойства StateImages, но лишь свойство StateIndex поддерживает изображение одного из первых 16 значков в списке. Значение индекса изображения состояния, превышающее число 15, все равно сводится к одному из заданных 16 изображений, а в программе для этого используется полученный от деления на 15 остаток.

```
if (stateindex >15)
    stateindex = stateindex % 15;
```

Обход дерева

Чтобы получить доступ к нужным данным в средстве просмотра дерева, приложение должно уметь реагировать на щелчки мышью или нажатия клавиш со стрелками на клавиатуре. Благодаря этому пользователь сможет просматривать, модифицировать, удалять или добавлять узлы дерева. Существует более 30 различных типов событий, которые можно обработать в классе TTreeView. Самыми популярными из них являются события OnClick и OnKeyUp.

Обработчик события OnClick отвечает на щелчок мышью, сделанный на элементе управления TreeView. Как показано в листинге 30.32, обработчик события OnClick используется для извлечения надписи, соответствующей определенному узлу дерева, и заполнения других элементов управления на панели приложения.

Листинг 30.32. Выбор узлов дерева

```
void __fastcall TFormFamilyTree::TreeViewFamilyClick(TObject *Sender)
{
    if (!TreeViewFamily->Selected)
    {
        if (TreeViewFamily->Items->Count == 0)
        {
            return;
        }
        TTreeNode *TopNode = TreeViewFamily->Items->GetFirstNode();
        TopNode->Selected = true;
    }
    ReflectNodeLabel(TreeViewFamily->Selected);
}
```

ReflectNodeLabel() — нестандартный метод, который передает надпись узла строке редактирования. Текст этого метода приведен на прилагаемом к книге компакт-диске (в папке Trees). Поскольку при обходе дерева используются клавиши со стрелками, указывающими направления перехода (вверх, вниз, влево, вправо), необходимо позаботиться об обработке событий OnKeyUp, чтобы обеспечить выделение (и выбор) соответствующего узла дерева. В обработчике события OnKeyUp организован перехват нажатий клавиш со стрелками.

```
void __fastcall TFormFamilyTree::TreeViewFamilyKeyUp(
                  TObject *Sender, WORD &Key, TShiftState Shift)
{
    if ((Key == VK_UP) || (Key == VK_DOWN) || (Key == VK_RIGHT)||Key == VK_LEFT)
        TreeViewFamilyClick(Sender);
}
```

Обратите внимание, что обработчик события `OnClick` вызывается в обработчике события `KeyUp`, когда нажатая клавиша ("в лице" переменной `key`) идентифицируется как одна из клавиш со стрелками. Нет никакой необходимости дважды использовать один и тот же код, и поскольку обработчик события `OnClick` обеспечивает выбор узла дерева, логично вызывать его именно при таких обстоятельствах.

Доступ к узлам дерева

В описанных выше примерах навигации по узлам дерева представлены способы получения доступа к этим узлам в средстве просмотра дерева. Для описания и определения отдельного узла дерева используется Borland-класс `TTreeNode`. Для идентификации узла дерева можно использовать ряд свойств (и подсвойств) класса `TTreeView`.

```
TTreeNode* node = TreeViewFamily->Selected;
TTreeNode* node = TreeViewFamily->TopItem;
TTreeNode* node = TreeViewFamily->Items->Item[x];
```

Свойство `Selected` используется для идентификации узла дерева, выбранного в данный момент. Использование свойства `Selected` иллюстрируется в методе обработки события `OnClick`.

Для отыскания первого узла в дереве можно использовать свойство `TopItem`.

Свойство `Items->Item[]` используется для получения доступа к другим узлам дерева. Как описано выше, свойство `Items` определяется в классе `TTreeNodes`, а подсвойство `Item[]`, определяемое свойством `Items`, представляет массив `TTreeNode`, содержащий все определенные узлы дерева. Зная абсолютную индексную позицию в дереве, с помощью этого массива можно получить доступ к любому узлу. Например, абсолютная индексная позиция первого узла дерева всегда равна 0 . Следовательно, доступ к вершине дерева реализуется с помощью выражения `Item[0]`. Абсолютную позицию других узлов можно найти посредством свойства `AbsoluteIndex`, также определенного в классе `TTreeNode`.

Однако существует опасность несоответствия узла своему прежнему индексу. Абсолютный индекс узла — величина *переменная*. Если дерево изменится (например, к нему будут добавлены новые узлы), узел, ранее расположенный в абсолютной позиции `x`, теперь может иметь индексное значение `y`. Поэтому лучше всего отслеживать местонахождение узла путем объявления указателя на переменную типа `TTreeNode` (именуемого ссылочным узлом) с последующим присвоением ему существующего узла дерева.

```
TTreeNode *rememberthisnode = TreeViewFamily->Selected;
```

В реализации ссылочного узла нет необходимости; этот узел дерева уже создан. Присвоение узла позволяет узнать его местоположение в памяти. Если узел дерева не будет удален, его адрес не изменится, в то время как абсолютный индекс может измениться. Чтобы узнать, не удален ли ссылочный узел, его можно проверить на равенство значению `NULL`.

```
if (!rememberthisnode) return;
```

Кроме того, очень важно иметь доступ к данным узла. Ниже показано, как реализуется считывание данных (значения свойства Data), содержащихся в узле дерева:

```
TPersonalInfo *info;
info = (TPersonalInfo*)(rememberthisnode->Data);
```

Следует иметь в виду, что свойство Data узла дерева определяется void-параметром, который позволяет присоединять любые данные (как показано в примере функции AddChildObject()). Следовательно, чтобы считать данные в нужном формате, необходимо к ожидаемой структуре указателя применить операцию приведения типа.

Поиск данных

По мере "разростания" дерева поиск элементов данных затрудняется. Пользовательская функция поиска данных (представленная в листинге 30.33) демонстрирует способ определения местонахождения узла дерева, основанный на обнаружении надписи узла.

Листинг 30.33. Поиск надписи узла дерева

```
int__fastcall FindTreeNodeBasedOnLabel(TTreeNodes*Tree, AnsiString SearchString)
{
    TTreeNode *CurItem = Tree->GetFirstNode();
    if (CurItem) do
    {
        AnsiString title = CurItem->Text;
        if (AnsiCompareStr(title, SearchString) == 0)
        {
            return CurItem->AbsoluteIndex;
        }
        CurItem = CurItem->GetNext();
    } while (CurItem);
    return -1; // Поиск оказался неудачным.
}
```

Эта функция просто опрашивает все узлы, идентифицируемые указателем на класс TTreeNodes, до тех пор, пока не обнаружит совпадения метки с заданной строкой. Это — абстрактная процедура, которая принимает указатель элемент управления TTreeNodes. Пример использования этой функции можно найти на прилагаемом к книге компакт-диске.

Эта функция также способна найти узел Node на основе информации об объекте, которая может содержаться в объекте Data узла Node. Пользовательская функция поиска данных (представленная в листинге 30.34) демонстрирует способ определения местонахождения узла дерева, основанный на обнаружении уникального идентификатора ID параметра Data. (Примечание: ID — это свойство структурного типа для элемента Data, использование которого показано в проекте famtree.bpr, находящемся на прилагаемом к книге компакт-диске.)

Листинг 30.34. Поиск значения ID узла дерева

```
int __fastcall FindTreeNodeBasedOnID(TTreeNodes *Tree, unsigned int ID)
{
    TTreeNode *CurItem = Tree->GetFirstNode();
    if (CurItem) do
    {
```

```
        if (CurItem->Data)
        {
            if (PInfo(CurItem->Data)->ID == ID)
                return CurItem->AbsoluteIndex;
        }

        CurItem = CurItem->GetNext();
    }while (CurItem);
    return -1; // Поиск оказался неудачным.
}
```

Эта функция очень похожа на описанную выше функцию `FindNodeBasedOnLabel`, за исключением того, что она выполняет поиск уникального `ID`, содержащегося в объекте `Data`.

Отображение данных о количестве дочерних узлов

Иногда средство просмотра дерева необходимо обогащать некоторой статистической информацией, например отображать количество подузлов, содержащихся в узле (особенно в том случае, когда узел находится в свернутом состоянии). Такую дополнительную информацию можно отобразить в надписи самого узла. Функция, представленная в листинге 30.35, демонстрирует метод присоединения в конец надписи узла информации о количестве его дочерних узлов.

Листинг 30.35. Добавление в надпись узла количества его дочерних узлов

```
void __fastcall AddCountToNode(TTreeNodes*tree)
{
     tree->BeginUpdate();
     TTreeNode *CurItem = tree->GetFirstNode();
     while (CurItem)  // Обход всех узлов дерева.
    {
        CurItem->Count;
        AnsiString NewName = CurItem->Text;
        StripParenCount(NewName);

        if (CurItem->Count > 0)
            CurItem->Text = NewName + " (" + CurItem->Count +")";
          else
              CurItem->Text = NewName;
          CurItem = CurItem->GetNext();
     }
     tree->EndUpdate();
}
```

В этом примере используются методы `BeginUpdate()` и `EndUpdate()`, определенные в классе `TTreeView`, которые служат для ограничения и управления обновлением средства просмотра дерева. Функция `BeginUpdate()` запрещает обновление дерева, а функция `EndUpdate()` снова позволяет обновление. Подобно некоторым определенным выше пользовательским утилитам, функция `AddCountToNode()` опрашивает каждый узел, начиная с первого. Первый узел опрашивается с помощью метода `GetFirstNode()`, определенного в классе

TTreeView. Все последующие узлы считываются с помощью метода GetNext(). Свойство Count, определенное в классе TTreeNode, идентифицирует количество подузлов следующего уровня (дочерних узлов), содержащихся в узле. Значение этого свойства заключается в круглые скобки и добавляется в конец текстовой строки узла.

В листинге 30.35 вы могли заметить обращение к функции StripParenCount(), которая используется для удаления текста о количестве подузлов, расположенного в конце надписи. Код этой функции представлен в листинге 30.36.

Листинг 30.36. Удаление текста о количестве подузлов из надписи узла

```
bool __fastcall StripParenCount(AnsiString &text)
{
    AnsiString NewName;
    bool result = true;
     int location = AnsiPos(" ((", text);
     if (location > 0)
     {
          NewName = text.SubString(0, (location-1));
          result = true;
     }
     else // Нечего стирать.
    {
        result = false;
        return result;
    }
    text = NewName;
    return result;
}
```

Перемещение узлов вверх и вниз

Часто некоторые действия не обеспечиваются VCL-методами — например, когда пользователю нужно переместить узлы дерева вверх или вниз.

В программе, приведенной на прилагаемом к книге компакт-диске, предусмотрено, что при щелчке пользователем правой кнопкой мыши на узле дерева появляется контекстное меню. Такие меню назначаются средству просмотра дерева с помощью свойства PopUpMenu, что дает возможность пользователю выполнять различные действия с выделенными данными. К подобным действиям относится перемещение узлов дерева вверх и вниз. Для поддержки этой возможности и была создана утилита, представленная в листинге 30.37.

Листинг 30.37. Перемещение узла дерева вверх

```
void __fastcall MoveUpTree(TTreeNode *item)
{
    if (item = NULL)
    {
        MessageBeep(MB_OK);
        return;
    }
    TNodeAttachMode AttachMode = naInsert;
```

```
    TTreeNode *ppItem = item->getPrevSibling();
    int level = item->Level;
    if (ppItem == NULL)
    {
        MessageBeep(MB_OK);
        return;
    }
    if (ppItem-> Level != level)
    {
        MessageBeep(MB_OK);
        return;
    }
    item->MoveTo(ppItem, AttachMode);
}
```

В качестве параметра этой функции передается указатель на объект класса `TTreeNode`. Метод `getPrevSibling()`, определенный в классе `TTreeNode`, предназначен для опроса предыдущего узла на текущем уровне древовидной структуры. Если предыдущий "братский" узел существует, нужный узел вставляется перед этим "братским" узлом, по сути перемещая его вверх.

В следующем фрагменте кода показано, как эта утилита используется в программе.

```
void __fastcall TFormFamilyTree::ActionNodeMoveUpExecute(TObject *Sender)
{
    MoveUpTree(TreeViewFamily->Selected); // Перемещение элемента вверх.
}
```

Не внося никаких изменений, функцию `MoveUpTree()` также можно использовать в программе для перемещения узла вниз.

```
void __fastcall TFormFamilyTree::ActionNodeMoveDownExecute(TObject *Sender)
{
    // Для перемещения нужного элемента вниз достаточно
    // переместить вверх нижний элемент.
    TTreeNode *pItem = TreeViewFamily->Selected->getNextSibling();
    if (pItem != NULL)
        MoveUpTree(pItem);
    else
        MessageBeep(MB_OK);
}
```

Как показано в этом примере, следующий узел (расположенный под выделенным в данный момент узлом) считывается с помощью метода `getNextSibling()` и передается в качестве параметра функции `MoveUpTree()`. Перемещение вверх следующего "братского" узла создает иллюзию, что выбранный узел перемещается вниз.

Выполнение операции „перетащить и опустить“

Возможность перетаскивать с помощью мыши элементы узлов необходима во многих приложениях, которые поддерживают средства просмотра деревьев. Существует множество аспектов операции "перетащить и опустить", которые стоило бы рассмотреть, и эта тема

вполне “потянула” бы на целую главу. Однако не будем пытаться объять необъятное и рассмотрим конкретный пример.

Программа использования средства просмотра дерева (она находится на прилагаемом к книге компакт-диске) позволяет пользователю переместить любой узел с одной позиции дерева на другую. Для поддержки этого аспекта операции “перетащить и опустить” используются два события, определенные в классе TTreeView: OnDragDrop и OnDragOver.

Вот как выглядит обработчик события OnDragOver.

```
void __fastcall TFormFamilyTree::TreeViewFamilyDragOver(
                TObject *Sender, TObject *Source, int X, int Y,
                                 TDragState State, bool &Accept)
{
    if (Source->InheritsFrom(__classid(TTreeView)))
        Accept = true;     // Используем возможность перетаскивать
                           // выделенный узел.
}
```

Событие OnDragOver используется для определения того, подходит ли к перетаскиваемому объекту эта операция, т.е. является ли она приемлемой для данного объекта. Это самый простой обработчик событий из ряда *drag-and-drop*-обработчиков. Перетаскиваемый объект (он идентифицируется как Source) анализируется на предмет принадлежности классу TTreeView или его потомкам. Это реализуется посредством метода InheritsFrom(), определенного в классе TObject. Как упоминалось выше, класс TTreeView использует (и наследует) класс TTreeNode ради свойств Selected и Items->Item[]. Если условие InheritsFrom() соблюдено, параметру Accept задается значение true, что позволяет пользователю перетаскивать узел в пределах дерева.

Обработчик события OnDragDrop имеет более сложный механизм. В следующем примере функции (листинг 30.38) обработчик события OnDragDrop используется для организации действий, которые должны произойти, когда пользователь опускает перетаскиваемый элемент на другую позицию в том же дереве. Эта операция называется *самоуправлением дерева* (tree self-manipulation), поскольку в ней не участвуют никакие элементы управления, помимо средства просмотра дерева.

Листинг 30.38. Перетаскивание элементов в дереве

```
void __fastcall TFormFamilyTree::TreeViewFamilyDragDrop(
                 TObject *Sender, TObject *Source, int X, int Y)
{
    TTreeNode *pItem = TreeViewFamily->GetNodeAt(X,Y);
    if ((Source == (TObject *)TreeViewFamily) && (TreeViewFamily->Selected))
    {
        THitTests HT = TreeViewFamily->GetHitTestInfoAt(X,Y);
        TNodeAttachMode AttachMode;
        if (HT.Contains(htOnItem) || HT.Contains(htOnIcon))
            AttachMode = naAddChild; // Опускаемый узел был выбран.
        else if (HT.Contains(htNowhere))
            AttachMode = naAdd;      // Опускаемый узел не был выбран.
        else if (HT.Contains(htOnIndent))
            AttachMode = naInsert;  // Вставляем перед выбранным узлом.
        else
            return;
```

```
        TreeViewFamily->Selected->MoveTo(pItem, AttachMode);
    }
}
```

Параметр `Source` события `OnDragDrop` указывает на перетаскиваемый и опускаемый узел, а параметр `Sender` (как и в большинстве обработчиков событий VCL) — на элемент управления, включающий объект, который требует определенного управления. В данном случае элементом управления является средство просмотра дерева, включающее перетаскиваемый узел. Параметры `X` и `Y` метода `OnDragDrop` идентифицируют координаты указателя мыши, расположенного над позицией предполагаемого "приземления" объекта. Метод `GetNodeAt()` используется для считывания узла дерева, расположенного в позиции с координатами `X` и `Y`. А метод `GetHitTestInfoAt()` используется для получения более подробной информации о позиции с координатами `X` и `Y`, поскольку она относится к клиентской области средства просмотра дерева. В данном случае метод `GetHitTestInfoAt()` определяет, куда должен быть помещен перетаскиваемый узел. При этом возможны три варианта.

- Узел добавляется к существующему узлу в качестве дочернего.
- Узел вставляется перед существующим в качестве "братского".
- Узел добавляется в пределах клиентской области средства просмотра дерева как новый родительский узел (независимо от других узлов).

Наконец, метод `MoveTo()`, определенный в классе `TTreeNode`, используется для перемещения узла в новую позицию в пределах того же дерева.

Список этих примеров можно было бы расширить за счет перетаскивания данных "между" различными деревьями. Мы предлагаем читателю самостоятельно рассмотреть возможные пути решения этой задачи.

Модификация узла

После добавления узла важно обеспечить возможность его модификации. Вполне вероятно, что пользователь захочет изменить надпись узла или связанные с ним данные.

Модификация надписи узла

Как видно на левой панели окна Windows Explorer (Проводник Windows), существует два основных метода модификации надписи узла внутри дерева.

- Пользователь может выделить в дереве узел и щелкнуть левой кнопкой мыши второй раз (не путайте с двойным щелчком), выполнив так называемое "вторичное выделение" ("second select"). Это приведет к появлению рамки вокруг узла и мигающего курсора в конце текста. Мигающий курсор — это сигнал для пользователя, что он может добавить новый текст к существующей надписи узла либо удалить выделенный текст, заменив его новым.
- Пользователь может щелкнуть правой кнопкой мыши, открыв контекстное меню, содержащее различные команды, в том числе и команду Rename (Переименовать). При выборе команды Rename вокруг узла появляется рамка, а в конце текста — мигающий курсор. Далее пользователь может выполнить действия, описанные в первом методе модификации надписи узла.

Элемент управления `TTreeView` фирмы Borland обеспечивает автоматизированную поддержку первого метода (свойство `ReadOnly` компонента `TTreeView` в этом случае имеет зна-

чение false). При выполнении программы пользователь может "вторично выделить" узел дерева и переименовать его, как это делается в утилите Windows Explorer. Обработчик события OnEdited используется для "перехвата" процесса изменения текста надписи узла, чтобы его можно было отразить на панели, содержащей строки редактирования. Второй метод можно реализовать программным путем, перехватив вызов обработчика событий мыши (щелчок правой кнопкой мыши) и обеспечив отображение контекстного меню с выделенной командой модификации текста и последующий переход в режим редактирования.

```
void __fastcall TFormFamilyTree::TreeViewFamilyEdited(
                TObject *Sender, TTreeNode *Node, AnsiString &S)
{
    if (S.Length() > 0)
    {
        Node->Text = S;
        ReflectNodeLabel(Node);
   }
}
```

В этом примере, если длина модифицированной надписи допустима, свойство Text узла дерева модифицируется для отражения новой надписи.

Модификация объекта данных узла

Помимо модификации надписи узла, важно разрешить пользователю вносить изменения и в элементы Data, связанные с выделенным узлом. Как это делается, показано в листинге 30.39.

Листинг 30.39. Модификация элементов данных, связанных с узлом дерева

```
void __fastcall TFormFamilyTree::ActionNodeModifyExecute(TObject *Sender)
{
    TTreeNode *tempnode = TreeViewFamily->Selected;
    AnsiString text = CreateLabel(EditFirstName->Text, EditLastName->Text);
    // CreateLabel - это специальная процедура, предназначенная
    // для генерации текстовой надписи, которая может быть
    // использована для нового узла дерева.
    if (text.Length() == 0) // Проверка на неверное имя.
    {
        MessageBeep(MB_OK);
        return; // Имя не предоставлено, т.е. оно неверно, а на
                // "нет" и суда (т.е. действий) нет, поэтому -
                // выход из функции.
    }
    TreeViewFamily->Selected->Text = text;
    TPersonalInfo *info;
    if (tempnode->Data == NULL) // Для этого узла данные еще не установлены.
    {
        info = new TPersonalInfo;
        TransferPersonalInfoFromEditEntry(0, text, info);
        tempnode->Data = info;
    }
    else
```

```
    {
        info = (TPersonalInfo*)(tempnode->Data);
        // Приводим данные к указателю TInfo.
        TransferPersonalInfoFromEditEntry(0, text, info);
    }
    if (info->Gender)
    {
        tempnode->ImageIndex = gPersonMale;
        tempnode->SelectedIndex = gPersonMaleSelected;
    }
    else
    {
        tempnode->ImageIndex = gPersonFemale;
        tempnode->SelectedIndex = gPersonFemaleSelected;
    }
    AddCountToNode(TreeViewFamily->Items);
}
```

Эта программа позволяет пользователю модифицировать надписи узла на панели, содержащей набор строк редактирования.

Как показано в программе, при щелчке на кнопке Apply Change (Применить изменения) значения соответствующих строк редактирования, расположенных на панели в правой части окна программы, сцепляются (конкатенируются), и полученное значение заменяет старую надпись выделенного узла. Надпись узла изменяется при модификации значения свойства `Text`. Кроме того, с помощью пользовательской функции `TransferPersonalInfoFromEditEntry()` обновляются любые изменения в строках редактирования данных, относящихся к выделенному узлу (код этой функции можно найти на компакт-диске, прилагаемом к книге).

Удаление узла

Необходимо также предоставить пользователю возможность удалить отдельный узел или целую ветвь узлов. С этой задачей может справиться метод `Delete()`, определенный в классе `TTreeNode`.

```
node->Delete();
```

Однако в случае существования элемента `Data` его автоматического удаления внутри метода `Delete()` не происходит. Для удаления объекта данных требуются дополнительные нестандартные действия до вызова метода `Delete()`.

```
if (node->Data != NULL)
    delete[] node->Data;
node->Delete();
```

Если в программе предусмотрена возможность отмены удаления (команда Undo), элемент `Data` должен быть извлечен или даже добавлен, если он не существовал ранее, а затем отмечен для удаления.

```
TInfo *info;
if (node->Data)
    info = (TInfo*)(node->Data); // Извлекаем элемент данных.
else
{
```

```
    // Лучше добавить данные на случай, если пользователь отменит удаление.
    info = new TPersonalInfo;
    info->ID = GetNewID();    // Получаем уникальный id.
    info->Name = node->Text;
    node->Data = (void*)info; // Теперь добавляем его к узлу как объект.
}
info->deleted = true;         // Отмечаем его для удаления.
```

В этом фрагменте программы приведен пример организации "мягкого удаления". Ведь на самом деле узел не удаляется, поскольку вслед за удалением "спохватившийся" пользователь может выполнить команду отмены. Существует несколько способов реализации мягких удалений с целью поддержки отмены (Undo) и повторного выполнения (Redo), которые будут описаны в следующем разделе.

Для ответа на нажатие клавиши <Delete> можно использовать обработчик события `KeyDown`, определенный в классе `TTreeView`. Пример его использования показан в следующем фрагменте программы.

```
void __fastcall TFormFamilyTree::TreeViewFamilyKeyDown(
                  TObject *Sender, WORD &Key, TShiftState Shift)
{
    if (Key == VK_DELETE)
    {
        Key = NULL;
        ActionNodeDeleteExecute(Sender);
    }
}
```

В этой процедуре анализируется нажатая клавиша: а вдруг это как раз клавиша <Delete>? Если — да, делается обращение к обработчику события удаления. Однако перед его вызовом переменную `Key` нужно установить равной значению `NULL`, чтобы избежать случайных удалений при автоматическом выделении новых узлов.

Поддержка отмены удаления и его повторного выполнения

К сожалению, разработчики фирмы Borland не предусмотрели свойства `Hide` для узлов дерева. А оно могло бы пригодиться для поддержки операций отмены и повторного выполнения, которые реализованы во многих коммерческих программах. Если пользователь случайно удалил узел и хотел бы его "воскресить", свойство `Hide` решило бы эту тривиальную задачу. Если бы свойство `Hide` было доступно, то при выборе команды Undo (Отменить), программа просто сняла бы с "удаленного" узла "шапку-невидимку", сделав его видимым. Несмотря на это досадное упущение, существует по крайней мере два способа поддержки команд Undo и Redo для узлов древовидной структуры.

Первый метод

Проще всего при выборе пользователем команды удаления узла присвоить ему признак удаления. Этот узел в действительности не будет удален, а лишь помечен для удаления. Как упоминалось выше, такое удаление называется "мягким". Этот метод удаления используется во многих программах, например Lotus Notes. Пользователю предоставляется возможность

завершить удаление, выполнив команду Refresh или Update (Обновить). Один из возможных вариантов реализации метода мягкого удаления демонстрируется в листинге 30.40.

Листинг 30.40. Мягкое удаление узла дерева

```
void __fastcall TFormFamilyTree::ActionNodeDeleteSoftExecute(TObject *Sender)
{
    TTreeNode*parentnode;
    TTreeNode* node;
    node = TreeViewFamily->Selected;
    parentnode = TreeViewFamily->Selected->Parent;
    // Проверяем, существуют ли дочерние узлы...
    if (node->HasChildren)
    {
       // Уведомляем пользователя о существовании дочерних узлов.
          int response =
          Application->MessageBox("Are you sure you wish to delete?",
                                  "Node Has Children", MB_YESNO);
          if (response == IDNO)
          {
               MessageBeep(MB_OK);
               return;
          }
     }
     TPersonalInfo *info;
     info = (TPersonalInfo*)(node->Data);
     ActionEditUndo->Enabled = AddToUndoRedoList(
                                        TreeViewUndo->Items, info->ID,
                                        info, node->ImageIndex, true);
     node->OverlayIndex = gDeleted;

     if (parentnode)
          parentnode->Selected = true; // Выбираем родительский узел.
    else
    {
        ActionNodeModify->Enabled = false;
        ActionNodeDeleteSoft->Enabled = false;
    }
    TreeViewFamilyClick(Sender);
}
```

Чтобы показать, что узел недоступен для пользователя, используется изображение покрытия, которое накладывает растровый символ NOT поверх изображения узла. Кроме того, поддерживаемый узлом элемент Data содержит признак (флаг) свойства, именуемый deleted, который предназначен для внутренней идентификации факта удаления узла. Когда пользователь удаляет узел дерева, признак deleted отмечается значением true, и используется изображение покрытия.

Поскольку средство просмотра дерева представляет собой эффективный контейнер связного списка, подобные древовидные структуры можно использовать для управления Undo- и Redo-списков. Адрес мягко удаляемого объекта добавляется в древовидный Undo-список.

Позже, если пользователь решит отменить удаление узла с помощью команды Undo, осуществляется доступ к верхнему объекту, адресуемому древовидным Undo-списком, в результате чего его признак `deleted` устанавливается равным значению `false`, и ликвидируется покрывающее изображение. Затем "помилованный" узел добавляется в Redo-список (на случай, если пользователь все-таки решит удалить его). На прилагаемом к книге компакт-диске находится программа Family Tree, которая демонстрирует действие команд отмены и повторного выполнения, обеспечивая "невидимую" реализацию этих двух функций.

Второй метод

Второй способ поддержки средства Undo/Redo — использовать два дерева: главное и подчиненное. Мы не иллюстрируем эту идею программой, но это вполне осуществимый и практичный вариант. Подчиненное дерево создается во время разработки (помещается в форму), а главное дерево — либо во время выполнения приложения (как объект класса `TTreeNodes`), либо во время разработки как объект класса `TTreeView`, причем его свойство видимости устанавливается равным значению `false`. Главное дерево невидимо и используется для включения супермножества того, что реально видимо в подчиненном дереве. Подчиненное дерево также называется *пользовательским представлением*.

Идея этого метода — использовать свойство `Data` для каждого узла дерева. Для всех видимых узлов признак `deleted` имеет значение `false`. При удалении какого-либо узла дерева признак `deleted` для эквивалентного узла главного дерева устанавливается равным значению `true`. Как и в описанном выше первом методе, удаляемый объект добавляется в Undo-список. Затем выполняются следующие действия.

1. Для подчиненного дерева вызывается метод `BeginUpdate()`, чтобы временно предотвратить обновление отображения.
2. Для подчиненного дерева вызывается метод `Clear()`, определенный в классе `TTreeNodes`. При этом из дерева удаляются все надписи узлов.
3. Узлы главного дерева, у которых признак `deleted` равен значению `false`, копируются в очищенное (пустое) подчиненное дерево.
4. Для подчиненного дерева вызывается метод `EndUpdate()`. Все изменения в дереве становятся видимыми для пользователя.

Впоследствии, если пользователь выберет команду отмены, будет осуществлен доступ к верхнему объекту, адресуемому Undo-списком, в результате чего признак `deleted` устанавливается равным значению `false`. Затем повторяются действия, описанные в пп. 1–4. По завершении п. 4 ранее удаленный узел снова появляется в видимом компоненте дерева. Адрес объекта `Data`, удаление которого было отменено, помещается в вершину Redo-списка.

Если пользователь выберет команду Redo (снова удалить) для объекта узла, будет осуществлен доступ к верхнему объекту, адресуемому Redo-списком, в результате чего признак `deleted` устанавливается равным значению `true`. Затем повторяются действия, описанные в пп. 1–4. По окончании п. 4 создается впечатление, что ранее "воскрешенный" узел снова удаляется. Адрес повторно удаленного объекта `Data` помещается в вершину Undo-списка.

Большинство описанных выше действий выполняется благодаря тому, что главный, подчиненный, Undo- и Redo-списки могут ссылаться на один и тот же адрес памяти, содержащий структуру данных узла, и модифицировать ее элементы.

Хотя в этом примере программы для иллюстрации Undo- и Redo-процессов были использованы компоненты `TTreeView`, опускаемые в форму из палитры компонентов при разработке, ничто не мешает использовать компоненты `TTreeNodes` для управления Undo- и Redo-списками

при выполнении программы. Кроме того, какие бы данные ни удалялись из средства просмотра дерева или некоторого другого компонента (список или текст), для управления мягким удалением объектов различного типа можно успешно использовать Undo- и Redo-списки.

Сохранение дерева

До сих пор мы рассматривали, как добавлять узлы дерева, модифицировать их, перемещать, удалять и отменять операции с узлами в пределах одного дерева. Не говорили мы пока о том, как сохранить средство просмотра дерева. Разработчики фирмы Borland предусмотрели в классе `TTreeView` два метода, которые позволяют сохранить дерево: `SaveToStream()` и `SaveToFile()`. Метод `SaveToStream()` обеспечивает наибольшую гибкость, создавая возможность направлять данные дерева в поток. Поток может представлять различные формы средств сохранения информации, это может быть, например, динамическая память или дисковый файл. Метод `SaveToFile()`, чтобы выполнить задачу сохранения дерева в дисковом файле, в действительности внутренне использует метод `SaveToStream()`. Следовательно, чтобы сохранить дерево на диске, достаточно передать методу `SaveToFile()` допустимое имя файла (и путь). Ниже приводится пример сохранения дерева в заданном файле.

```
if (SaveDialog1->Execute())
    TreeViewFamily->SaveToFile(SaveDialog1->FileName);
```

Открытие существующего файла аналогично использованию Borland-метода `LoadFromFile()`, предусмотренного для компонента просмотра дерева.

```
if (OpenDialog1->Execute())
    TreeViewFamily->LoadFromFile(OpenDialog1->FileName);
```

Недостатком `SaveTo`-методов является то, что они не сохраняют информацию, содержащуюся в объекте данных узла. Для обеспечения этой возможности часто требуются нестандартные методы. Один из них — для сохранения данных в плоском файле применить доступный для всех анализатор, например анализатор XML. Преимущество XML-анализатора в том, что он естественным образом поддерживает иерархические взаимосвязи, подобные древовидной структуре. К сожалению, рассмотрение анализаторов выходит за рамки обсуждения `TTree`-компонентов, поэтому здесь не приводится программа использования анализаторов. Однако существует способ, предусматривающий включение методов в структуру данных, которая используется для определения элемента свойства `Data` для узла. Соответствующие вызовы утилиты анализатора можно встроить в эти методы. Один метод предназначен для записи проанализированных данных в строковый буфер, а другой — для чтения проанализированных данных из строкового буфера. Чтобы сохранить и открыть файлы, сохраненные на диске, эти методы должны вызываться внешней функцией в процессе обхода дерева.

Резюме по компоненту `TTree`

В этом разделе были рассмотрены возможности разработанного компанией Borland набора компонентов `TTree`, которые могут оказаться весьма полезными для C++-разработчиков. Вы познакомились с методами заполнения средства просмотра древовидной структуры и эффективного управления объектами данных, включающими различные способы навигации, поиска и доступа к информации, относящейся к узлам дерева, а также способы манипуляции узлами с помощью мыши (“перетащить-и-опустить”). Кроме того, здесь были рассмотрены возможности организации поддержки команд отмены и повторного выполнения (Undo и Redo) и сохранения дерева в дисковом файле. На компакт-диске,

прилагаемом к книге, находится ряд TTree-утилит, которые расширяют возможности оригинального компонента Borland и могут быть использованы в качестве основы для новых TTree-компонентов библиотеки VCL. Профессиональные разработчики по достоинству оценят могущество TTree-компонентов для эффективного управления элементами иерархических структур и объектов данных.

Реализация утилиты выделения пиктограмм

Как это ни парадоксально, но труднее всего в книге, посвященной C++Builder, найти способ изложения материала именно ввиду простоты использования C++Builder.

По сути, нам остается лишь перечислить действия в правильной последовательности: "сначала сделайте это..., затем переместите курсор и сделайте то...". Но вы ведь не считаете себя беспилотным аппаратом, бездумно выполняющим команды, а хотели бы научиться чему-нибудь полезному, не так ли? Например, вряд ли кто-либо усомнится в полезности пиктограмм, но ведь их так утомительно делать. Гораздо легче "позаимствовать" их у других программ.

В Internet есть узлы, которые специализируются на пиктограммах и растровых изображениях. Если вам не хватает таланта художника для собственноручного создания пиктограмм с нуля, можно "присмотреть" подходящий вариант на одном из таких узлов или затратить несколько минут для написания программы в среде C++Builder, позволяющей в некотором роде "заимствовать" пиктограммы.

На заметку

Проект IconBandit, который находится на компакт-диске, прилагаемом к книге (в папке IconBandit), имеет больше функций, чем приведенная здесь версия. В этом разделе рассматриваются основные методы, а программа на компакт-диске представляет собой законченное приложение, которое может служить ориентиром для создания собственных продуктов.

После запуска C++Builder вы обычно видите пустую форму перед пустой оболочкой программы. Предположим, вы хотите наделить свою программу специального назначения собственной пиктограммой, которая будет отображаться в нескольких местах: в левом верхнем углу формы, на панели задач в нижней части экрана и в окне утилиты Проводник Windows. Для установки пиктограммы выберите команду Project⇨Options (Проект⇨Параметры), чтобы открыть вкладку Application (Приложение) диалогового окна Options.

Щелкните на кнопке Load Icon (Загрузить пиктограмму), чтобы открыть меню File Open (Открыть файл). Выберите нужную пиктограмму и щелкните на кнопке Open (Открыть).

Когда вы сохраните проект (используйте для него имя IconBandit.bpr, а для формы — IBForm.cpp), будет создано несколько файлов. Один из них — файл ресурсов с расширением .RES. Файлы ресурсов могут содержать пиктограммы, растровые изображения, текст и даже двоичные данные. В данный момент нас интересуют только пиктограммы. Для нашей программы мы создали пиктограмму и внесли ее в проект IconBandit.bpr с помощью команды Project⇨Options⇨Application и щелчка на кнопке Load Icon. Конечным именем программы будет имя IconBandit.exe. Используя команду Tools⇨Image Editor (Сервис⇨Редактор изображений), можно запустить редактор изображений. В окне Image Editor выберите команду File⇨Open (Файл⇨Открыть), чтобы открыть .RES-файл для своего проекта. Теперь дважды щелкните на надписи Icon, чтобы появился элемент MAINICON. Дважды щелкните на этом элементе, чтобы открыть редактор для пиктограммы. Поскольку цветное окаймление пиктограммы прозрачно, на экране его не будет видно. Пиктограмма для этой программы показана на рис. 30.13.

Рис. 30.13. Пиктограмма IconBandit в окне Image Editor

На этом рисунке вы видите готовую к употреблению пиктограмму в `.RES`-файле. C++Builder автоматически ее обработает, "не вынимая из упаковки". Однако в `.RES`-файле необязательно должна содержаться только одна пиктограмма — их может быть там сколько угодно. Все они автоматически получают имена. Имя пиктограммы можно легко изменить, щелкнув на ней правой кнопкой мыши и выбрав из контекстного меню команду Rename (Переименовать).

Как пиктограмма `MAINICON`, так и другие пиктограммы, которые вы могли бы внести в этот `.RES`-файл, будут доступны во время работы программы. К ним можно получить доступ с помощью функции Win32 API `LoadIcon()`. Все API-функции Windows можно использовать в среде C++Builder, но есть более простой способ. Воспользуемся в нашей программе несколькими API-функциями и компонентами VCL. Кроме того, рассмотрим C++Builder-компонент `TActionList` и связанные с ним методы и события. Те, кто только начинает программировать в C++Builder, часто не обращают внимания на эти *списки действий* (Action Lists). И зря, поскольку они могут намного упростить процесс программирования.

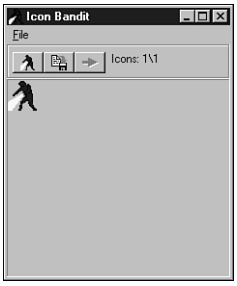


Рис. 30.14. Приложение Icon Bandit с захваченной пиктограммой

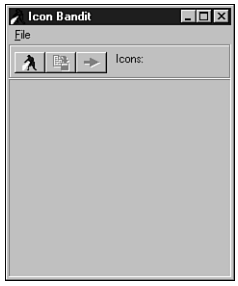


Рис. 30.15. Приложение Icon Bandit

Теперь посмотрим, как должна выглядеть законченная программа. Возможно, это будет неокончательная версия, поскольку вы наверняка захотите улучшить ее, обогатив новыми возможностями. "Лицо" этой маленькой программы показано на рис. 30.14 — так будет выглядеть результат нашей совместной работы. Задача программы — заглядывать в другие про-

граммы, загружать библиотеки и файлы пиктограмм, отображать пиктограммы, которые она способна увидеть, и по возможности записывать их на диск.

Обратите внимание, что программа имеет меню, хотя и совсем небольшое. Она оснащена также панелью инструментов с кнопками и текстом. А под панелью расположена область, предназначенная для отображения пиктограмм.

При самостоятельном запуске окно этой программы будет иметь вид, как на рис. 30.15.

Пиктограмма Icon Bandit видна сразу под панелью инструментов. Текст на панели содержит информацию о том, что это первая и единственная пиктограмма в программе.

На это стоит обратить внимание, поскольку, как отмечалось выше, в `.RES`-файле у вас может быть любое количество пиктограмм. При создании `.EXE`-файла `.RES`-файл связывается с вашей программой. Вышесказанное станет более ясным, если выполнить нашу программу, "нацелив" ее на файл `MORICONS.DLL`, находящийся в папке `Windows` или `Windows\System`.

Когда приложение Icon Bandit открывает программу или `.dll`-библиотеку, оно "пересчитывает" в ней количество пиктограмм. Если их обнаруживается более одной, то для действия `Next` свойство `Enabled` устанавливается равным значению `true`. Это позволяет выполнять прокрутку пиктограмм, обнаруженных в файле. Если вам понравится какая-либо из них, щелкните на кнопке Save (Сохранить), выберите из меню программы команду File⇨Save (Файл⇨Сохранить) или нажмите клавиши <Ctrl+S>.

Теперь вернемся назад и рассмотрим используемые в нашей программе компоненты и функции API.

Прежде всего опустите в форму компонент `TActionList` (Список действий), расположенный во вкладке Standard (Стандартная) палитры компонентов. После двойного щелчка на компоненте Action List открывается редактор Action (Действие). Мы собираемся создать три действия и использовать их вместо обработчиков событий в отдельной кнопке и меню обработчиков событий. Разместив в форме компонент Action List, перейдите на вкладку Win32 и расположите рядом с первым компонентом компонент `ImageList`. Дважды щелкните на компоненте `ImageList`, чтобы открыть диалоговое окно ImageList.

Щелкнув на кнопке Add (Добавить) и перейдя в папку Borland `Shared\Images\Buttons`, можете добавить значки для кнопки File Save и кнопки с изображением направленной вправо стрелки (для просмотра следующей пиктограммы). Из папки `IconBandit` (на компакт-диске, прилагаемом к книге) можете также добавить файл `IconBandit.ico`. После выполнения этих действий вам, возможно, будет предложено разделить изображение кнопки на два, поскольку выбранный вами размер изображения может не соответствовать требуемому. Ответьте на это предложение положительно (Yes). Загрузив изображения, вернитесь к списку действий (компоненту `Action List`). Щелкните на кнопке OK, а затем дважды щелкните на компоненте `Action List`.

На данном этапе для вас доступна лишь одна кнопка, именуемая New Action (Новое действие). Кнопка имеет комбинированный список, скрытый под расположенной рядом другой кнопкой с направленной вниз стрелкой. Этот список который можно использовать для выбора совершенно нового действия либо другого стандартного действия. *Стандартные действия* связаны с событиями из навигатора набора данных (dataset navigator) или окна редактирования. Они могут также обрабатывать сообщения процедур Window (например, свернуть окно или расположить окна каскадом). Теперь щелкните на кнопке New Action, чтобы добавить новое действие и установить следующие важные свойства.

1. Установите свойство `Caption` равным значению `&Open`.
2. Установите свойства `ImageIndex` и `ShortCut` равными нулю.
3. Установите для свойства `Category` значение `File`. Тем самым настраивается инструмент группирования, который сохранит аналогичные групповые действия в редакторе.

После изменения свойства Category легко заметить, что на панели Categories редактора Action List Editor появилась кнопка File. Щелкните на ней, чтобы установить ее свойства.

Закройте редактор Action List Editor и установите для свойства Image действия Action1 (из списка TActionList) значение ImageList1. Дважды щелкните на действии Action1 компонента TActionList, чтобы снова открыть редактор Action List Editor. Затем выделите категорию File. Обратите внимание на правую сторону (Actions), где заголовки (Captions) в данный момент отсечены. Пустое пространство отведено для изображений. Пока свойство ImageIndex имеет стандартное значение (–1), ни одного изображения не видно. Если изменить свойство ImageIndex, появится изображение, адресуемое этим свойством. Установите свойство ImageIndex так, чтобы оно ссылалось на заголовок нужного изображения. Для действия Action1 предназначена пиктограмма Open.

Необходимо убедиться в отсутствии ненужных изображений в списке ImageList. Если удалить их позже, то не исключено, что свойства ImageIndex, установленные вами в списке TActionList, будут указывать на несоотвествующее изображение.

А теперь создадим еще два действия для кнопок Save и Next. Для них можно оставить имена Action2 и Action3. Свяжем эти действия с изображениями с помощью свойства ImageIndex и назначим для каждого из них клавишу ускоренного доступа.

Поместите в форму панель (компонент TToolBar), расположенную во вкладке Win32 палитры компонентов. Если ее разместить у верхней границы формы, она сама выровняется вдоль этой границы. Или же установите для свойства выравнивания панели значение alTop. Щелкните правой кнопкой мыши на панели и выберите из контекстного меню команду New Button (Новая кнопка). Новая кнопка должна тут же появиться на панели. В окне Object Inspector поместите в вершину списка элемент для свойства Action. Выберите первое действие Action1. Вы поняли, что произошло? C++Builder просто автоматически заполнил значения других элементов. Эта кнопка теперь должна иметь изображение, нарисованное на ней, а событие OnButtonDown этой кнопки должно получить значение Action1Execute. Давайте-ка попытаемся сделать то же самое с меню.

Опустите в форму компонент Menu, а затем дважды щелкните на нем. Откроется редактор меню. От вас требуется теперь лишь ввести заголовки. Не забывайте при этом предварять каждую клавишу ускоренного доступа символом амперсанда (&). В процессе ввода можно использовать редактор свойств для отдельных элементов меню.

Как в случае с кнопками, установите свойство Action в вершину списка для действия, которое вы хотите связать с элементом меню. Если сейчас в окне Object Inspector выбрать вкладку Events, нетрудно заметить, что событие OnClick автоматически примет значение ActionNExecute.

Теперь нужно предусмотреть пространство для отображения пиктограмм. Из вкладки Additional палитры компонентов в верхнюю часть формы поместите компонент Image. Установите для свойства выравнивания значение alClient. Затем опустите в форму второй компонент Image и установите для его свойства выравнивания значение alClient, а для свойства Visible — значение false. Даже если вы не предусматриваете его отображение, все равно загрузите какой-нибудь растр, дважды щелкнув на свойстве Picture. Это растровое изображение будет служить фоном, чтобы создать эффект отсутствия каких-либо пиктограмм (для этого подойдет файл растрового изображения Background.bmp, который можно найти в папке IconBandit на компакт-диске, прилагаемом к книге).

Из вкладки Dialogs (Диалоговые окна) палитры компонентов опустите в форму диалоговые окна Open и Save. Теперь, наконец, можно приступать к программированию, хотя

C++Builder настолько облегчает работу программиста, что она не потребует от вас (по крайней мере в данном случае) больших усилий.

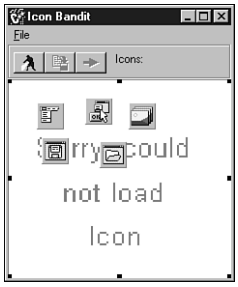


Рис. 30.16. Форма приложения Icon Bandit в режиме разработки

Если вы следовали нашим указаниям, ваша форма должна выглядеть примерно так, как показано на рис. 30.16.

Процедуре инициализации формы вообще не нужен никакой код. Однако нам придется добавить в класс формы две переменные. Для этого можно щелкнуть правой кнопкой мыши на файле `IBForm.cpp`, чтобы отобразить контекстное меню, предлагающее открыть исходный или заголовочный файл (Source/Header file). Именно так и поступите, а затем найдите раздел `private`. Теперь добавьте две длинные целочисленные переменные, как показано ниже.

```
private: // Объявления пользователя.
    long Index;
    long Limit;
```

Снова откройте редактор действий (для этого щелкните правой кнопкой мыши на компоненте `ActionList1`, выберите из меню команду Action List Editor) и выделите действие `Action1`. В окне Object Inspector перейдите на вкладку Events и дважды щелкните на элементе `OnExecute`. Тем самым вы создадите метод с именем `Action1Execute()`. Основная часть нашей программы будет состоять из этого метода.

Нам также потребуется вскоре установить еще несколько свойств, но пока рассмотрим процедуру, представленную в листинге 30.41.

Листинг 30.41. Метод `Action1Execute`

```
void __fastcall TForm1::Action1Execute(TObject *Sender)
{
    HICON I;
    Index = 0;
    if (OpenDialog1->Execute())
    {
        Limit = (long)ExtractIcon(HInstance, OpenDialog1->FileName.c_str(), -1);

        if (Limit > 1)
            Action3->Enabled = true;
        else
            Action3->Enabled = false;

        I = ExtractIcon(HInstance, OpenDialog1->FileName.c_str(), Index);
        if ((int)I > 1)
        {
            Image1->Picture->Icon->Handle = I;
            Action2->Enabled = true;
        }
        else
        {
            Image1->Picture->Assign(Image2->Picture);
            Action2->Enabled = false;
        }
    }
}
```

Сначала объявляется временная переменная I в качестве дескриптора для пиктограммы. Длинная целочисленная переменная Index, которую мы создали в закрытом (private) разделе класса формы, устанавливается равной нулю. Она нам вскоре потребуется.

Следующая (после инициализации переменной Index) строка программы несколько проблематична. Мы действительно поместили в форму диалоговое окно Open, и оно, даже если ничего не менять, будет вполне работоспособным. Однако мы можем облегчить жизнь нашим пользователям, установив свойство Filter этого диалогового окна. В окне Object Inspector можно либо дважды щелкнуть на свойстве Filter, либо щелкнуть на кнопке с троеточием, расположенной справа. При этом будет активизирован редактор фильтров (Filter Editor). В верхнюю строку столбца Filter Name введите комментарий All Icons followed by the file extensions in parentheses (Для всех пиктограмм расширения файлов заключены в круглые скобки). В столбце Filter введите расширения файлов, разделенные точкой с запятой. Это предоставит нашим пользователям максимально возможный выбор файлов. Затем введите отдельные типы файлов и их расширения. Это позволит пользователям ограничить типы файлов, просматриваемые в диалоговом окне Open. Завершив настройку, щелкните на кнопке OK.

В следующей строке программы вызывается функция ExtractIcon(). Мы передаем имя файла, возвращаемое при выборе пользователем файла, который необходимо открыть, экземпляр приложения (Application Instance) и индекс, равный –1, который означает специальную инструкцию для API. Тем самым вы уведомляете API-интерфейс, что вас интересует количество пиктограмм, содержащихся в файле. Обратите внимание, что мы приводим результат к типу long во избежание предупреждающих сообщений компилятора.

На заметку Обратите внимание, что, создавая сообщение, которое должно быть размещено на панели инструментов, мы использовали четыре отдельные командные строки, поскольку компилятор слишком разборчив в отношении операций конкатенации и символов "\\".

Обратите также внимание, что переменная Index содержит количество пиктограмм, которое мы собираемся отобразить, а переменная Limit — общее число пиктограмм в файле. Если значение Limit больше 1, мы разрешаем выполнение действия Action3, поэтому пользователь может прокручивать пиктограммы; в противном случае мы блокируем это действие.

При повторном вызове той же API-функции мы передаем ей в качестве третьего параметра вместо числа –1 значение переменной Index. На этот раз нам не нужно применять к результату операцию приведения типа, поскольку мы используем тип HICON. Если при возврате из функции оказывается, что переменная I больше 1 (т.е. пиктограмма обнаружена), то для отображения этой пиктограммы мы устанавливаем свойство Image1->Picture->Icon->Handle равным значению I. Кроме того, для действия Action2 мы устанавливаем свойство Enabled равным значению true, поэтому пользователь может сохранить эту пиктограмму на диске. Если в исследуемом файле не будет обнаружено ни одной пиктограммы, мы устанавливаем свойство Image1->Picture равным значению свойства Image2->Picture, тем самым отображая фоновый растр. Для действия Action2 мы также устанавливаем свойство Enabled равным значению false, поскольку не хотим, чтобы пользователь имел возможность сохранять несуществующую пиктограмму.

Эта часть программы совсем проста. Она содержит лишь несколько обращений к API-функциям и несколько операций присвоения значений переменным. Не следует думать, что теперь вас ожидают сплошные трудности. Остальная часть программы еще проще. Для подтверждения сказанного рассмотрим, что следует выполнить для сохранения пиктограммы на диске.

```
void __fastcall TForm1::Action2Execute(TObject *Sender)
{
    if (SaveDialog1->Execute())
    {
        Image1->Picture->Icon->SaveToFile(SaveDialog1->FileName);
    }
}
```

Этот фрагмент программы показывает: если пользователь выбрал имя и местоположение файла, мы должны обеспечить возможность сохранения файла `Image1->Picture->Icon` на диске. Подобно работе с диалоговым окном Open, следует установить фильтр и в этом диалоговом окне. Поскольку мы имеем дело только с пиктограммами, сохранение файла занимает лишь одну строку программы.

В окне Object Inspector устанавливаем для свойства `FileName` диалогового окна `SaveDialog1` значение `*.ICO`, чтобы пользователь видел все пиктограммы в любой выбранной им папке.

Процедура просмотра пиктограмм (приведенная в листинге 30.42) представляет собой вариацию процедуры, рассмотренной выше.

Листинг 30.42. Метод `Action3Execute()`

```
void __fastcall TForm1::Action3Execute(TObject *Sender)
{
    HICON I;
    if (Index < (Limit - 1))
        Index++;
    else
        Index = 0;

    I = ExtractIcon(HInstance, OpenDialog1->FileName.c_str(), Index);
    if ((int)I > 1)
    {
        Image1->Picture->Icon->Handle = I;
        Action2->Enabled = true;
    }
    else
    {
        Image1->Picture->Assign(Image2->Picture);
        Action2->Enabled = false;
    }
}
```

Мы по-прежнему устанавливаем переменные типа `HICON` и `AnsiString` (для сообщения). Однако здесь мы не запрашиваем количество пиктограмм. Нам это значение уже известно. Поэтому процедура начинается со сравнения индекса с предельным значением. Если индекс меньше единицы, мы инкрементируем его. В противном случае он устанавливается равным нулю.

Процесс “реанимации” индекса выполняется каждый раз, когда пользователь передает последнюю пиктограмму. С этого момента вторая процедура идентична первой.

Полный C++Builder-проект этой программы находится на компакт-диске (в папке `IconBandit`), прилагаемом к книге.

Выше высказывалось предположение, что вам обязательно захочется усовершенствовать эту программу. Например, дать возможность пользователю просматривать пиктограммы не только в прямом, но и в обратном направлении. Кроме того, пользователю было бы интересно знать размер каждой пиктограммы в пикселях. Воспользуйтесь приведенным здесь программным кодом и улучшите его — именно так становятся хорошими программистами!

Создание приложения, подобного Windows Explorer

В этом разделе мы будем использовать компонент `TTreeView` для создания приложения, отображающего список всех папок в системе Windows, включая и виртуальные папки (например, `My Computer`). Список откроет папка рабочего стола (Desktop), а само приложение будет выглядеть подобно утилите Проводник Windows (Windows Explorer).

Интерфейсы и функции оболочки Windows

В утилите Проводник Windows используются функции и интерфейсы оболочки. Вы могли бы подумать, что все дело в использовании функций, подобных `FindFirst()` и `FindNext()`. К сожалению, это не так. Если вы попытаетесь использовать эти функции для получения системных папок, то убедитесь в том, что они не в состоянии прочитать все папки, отображаемые Проводником Windows. Такие папки, как `My Computer` (Мой компьютер), `Recycle Bin` (Корзина), `Network Neighborhood` (Сетевое окружение) и `Printers` (Принтеры), им просто "не по зубам". Дело в том, что функции `FindFirst()` и `FindNext()` не могут обнаружить несуществующие папки. (Вернее, они существуют, но не поддерживаются системой.)

Эти специальные папки называются *виртуальными*. Они поддерживаются *пространством имен оболочки* (shell namespace). Оболочка использует Desktop (Рабочий стол) в качестве корня для объектов всех типов (файлов, принтеров, запоминающих и сетевых устройств).

Папка оболочки может представлять собой обычную папку системных файлов или специальную папку (подобно папкам `Desktop` или `Printers`). Кроме того, файл оболочки может быть реальным файлом (например, текстовым) или объектным файлом (например, приложение панели управления).

Каждая папка оболочки реализует интерфейс `IShellFolder`. Манипулировать папкой оболочки можно, считав ее интерфейс `IShellFolder`. Считывание интерфейса папки оболочки называется *связыванием с папкой*.

Для связывания с любой папкой (т.е. для считывания интерфейса `IShellFolder` папки) необходимо сначала осуществить связывание с папкой `Desktop`. Дело в том, что папка `Desktop` — корневая папка оболочки, а на все остальные папки используются ссылки относительно папки `Desktop`. Связывание с папкой `Desktop` реализуется с помощью функции `SHGetDesktopFolder`. После считывания интерфейса `IShellFolder` папки `Desktop` можно подключиться к любой ее подпапке, используя функцию-член `IShellFolder::BindToObjects()`. С помощью функции-члена `IShellFolder::EnumObjects()` можно также получить перечисление всех подпапок. Эта функция возвращает указатель на интерфейс `IEnumIDList`. Этот интерфейс можно использовать для опроса значений ID всех папок посредством функции `IEnumIDList::Next()`, которая возвращает указатель на объект, именуемый *PIDL* (*p*ointer to an item *id*entifier *l*ist — указатель на список идентификаторов).

Каждый объект оболочки имеет идентификатор элемента и присваиваемый ему PIDL. Идентификатор элемента уникально идентифицирует объект внутри родительской папки. PIDL-указатель уникально идентифицирует объект внутри оболочки.

PIDL-указатель — это ключевое значение для считывания всех видов полезной информации. Например, это значение можно передать функции оболочки `SHGetFileInfo()`, чтобы получить данные о файле (имя, местоположение и пиктограмма).

PIDL выделяется с помощью функции оболочки `SHGetMalloc()`, которая считывает интерфейс `IMALLOC`. `IMALLOC` имеет несколько методов, но один из них важнее остальных — это функция-член `IMALLOC::Free()`, используемая для освобождения памяти оболочки, на которую указывает PIDL-указатель.

Решение

Чтобы глубже понять то, о чем шла речь в предыдущем разделе, предлагаем создать новый проект, демонстрирующий возможность создания средства просмотра древовидной структуры, подобной дереву Windows Explorer. Единственное отличие рассматриваемого здесь проекта от Windows Explorer состоит в том, что в его дереве отображаются не только папки, но и файлы.

Поэтому, не тратя лишних слов, начнем с создания нового приложения. Поместите в форму следующие компоненты и установите их свойства, как показано в табл. 30.8. Не забудьте, что компоненты `TTreeView` и `TImageList` находятся во вкладке Win32 палитры компонентов.

Таблица 30.8. Компоненты и их свойства для проекта WExplorer

Компонент	Свойство	Значения
`TForm`	`Name`	`WinExplorer`
`TTreeView`	`Name`	`TV`
	`Align`	`alClient`
	`ReadOnly`	`true`
`TImageList`	`Name`	`ImageList1`
	`ShareImages`	`true`
`TButton`	`Name`	`Button1`
	`Caption`	`Show List`

Судя по содержимому приведенной выше таблицы, компоновка проекта довольно проста, но основная "жизненная сила" этого проекта заключена в функциях, определенных в заголовочном файле, как показано в листинге 30.43.

Листинг 30.43. Заголовочный файл проекта WinExplorer

```
#ifndef Unit1H
#define Unit1H
//---------------------------------------------------------------
#include <Classes.hpp>
#include <Controls.hpp>
#include <StdCtrls.hpp>
#include <Forms.hpp>
```

```
#include <ComCtrls.hpp>
#include <ImgList.hpp>
#include "shlobj.h"
#include "shellapi.h"
#include <ComCtrls.hpp>
#include <ImgList.hpp>
#include <FileCtrl.hpp>
#include <ActiveX.hpp>
//---------------------------------------------------------------
class TWinExplorer :public TForm
{
__published:    // Компоненты, управляемые IDE.
    TButton *Button1;
    TTreeView *TV;
    TImageList *ImageList1;
    void __fastcall Button1Click(TObject *Sender);
    void __fastcall TVCustomDrawItem(TCustomTreeView *Sender,
                    TTreeNode *Node, TCustomDrawState State,
                    bool &DefaultDraw);
    void __fastcall FormCreate(TObject *Sender);
    void __fastcall TVExpanding(TObject *Sender,
                              TTreeNode *Node, bool &AllowExpansion);
    String __fastcall StrRetToStr(LPITEMIDLIST pidl, LPSTRRET lpStr);
    void __fastcall AddNode(TTreeNode *Node, IShellFolder*Folder,
              String FolderName, String Path, LPITEMIDLIST pidl);
    int __fastcall GetImageIndex(AnsiString aUrl,
                               AnsiString APath, LPITEMIDLIST pidl);
    void __fastcall GetObjects(TTreeNode *Node);
    void __fastcall DrawButton(TTreeNode *Node, TRect ARect,
                                    int AWidth, TColor AColor);
    void __fastcall DrawLine(TCanvas *ACanvas, int x1, int y1, int x2, int y2);
private:    // Определения пользователя.
    typedef struct MyRec
    {
        bool FirstExpand;
        IShellFolder* ishlf;
    } TMyRec;
    typedef TMyRec*PMyRec;

public:    // Определения пользователя.
    __fastcall TWinExplorer(TComponent*Owner);
};
//---------------------------------------------------------------
extern PACKAGE TWinExplorer *WinExplorer;
//---------------------------------------------------------------
#endif
```

Объявленные в заголовочном файле проекта WinExplorer (см. листинг 30.43) основные его функции перечислены в табл. 30.9 с указанием назначения и источника вызова.

Таблица 30.9. Основные функции проекта WinExplorer

Функция	Назначение	Источник вызова
`StrRetToStr()`	Преобразует формат `SetRet` в строку	Обработчик события `OnClick` кнопки Button1 и функция `GetObjects()`
`AddNode()`	Добавляет узел к дереву	Обработчик события `OnClick` кнопки Button1 и функция `GetObjects()`
`GetObjects()`	Считывает объекты текущего узла	Обработчик события `OnExpanding` дерева
`GetImageIndex()`	Возвращает индекс изображения текущей папки или файла	Функция `AddNode()`
`DrawButton()`	Изображает кнопки дерева	Обработчик события `OnCustomDrawItem` дерева
`DrawLine()`	Рисует линии	Функция `DrawButton()`

Этот проект начинает свою работу со считывания списка системных изображений, который содержит все пиктограммы системных файлов. Дескриптор этого списка возвращает функция `SHGetFileInfo()` благодаря тому, что в качестве соответствующего параметра-флага передано значение `SHGFI_SYSICONINDEX`. Содержимое листинга 30.44 помещается в обработчик события `OnCreate` формы, чтобы дескриптор системных изображений можно было получить при запуске приложения.

Листинг 30.44. Считывание списка системных изображений и внесение его в список ImageList1

```
void __fastcall TWinExplorer::FormCreate(TObject *Sender)
{
    SHFILEINFO sfi ;
    unsigned long handle ;
    ImageList1->DrawingStyle = dsNormal;
    ImageList1->Height = 16;
    ImageList1->Width = 16;
    ImageList1->ShareImages = true;
    // Получение дескриптора системных изображений.
    handle = SHGetFileInfo("", 0, &sfi, sizeof(sfi),
                           SHGFI_SYSICONINDEX | SHGFI_SMALLICON);
     if (handle)ImageList1->Handle = handle;
     // Список ImageList1 содержит все системные пиктограммы.
}
```

При щелчке на кнопке Button1 приложение отобразит папку Desktop (подобно Windows Explorer). В обработчике события `OnClick` кнопки Button1 (как показано в листинге 30.45) приложение считывает интерфейс `IShellFolder`, который используется для добавления к дереву узла `Desktop` (корня).

Листинг 30.45. Обработчик события `OnClick` для кнопки Button1

```
//---------------------------------------------------------------
void __fastcall TWinExplorer::Button1Click(TObject *Sender)
```

```
{
    IShellFolder* ishellfolder;
    STRRET Path, Name;
    LPITEMIDLIST lpitemidlist;

    // Сначала читаем данные о местоположении папки Desktop.
    SHGetSpecialFolderLocation(Handle, CSIDL_DESKTOP, &lpitemidlist);
    // Получаем интерфейс IShellFolder папки Desktop.
    SHGetDesktopFolder(&ishellfolder);
    // Получаем ее имя и путь.
    ishellfolder->GetDisplayNameOf(NULL, SHGDN_NORMAL, &Name);
    ishellfolder->GetDisplayNameOf(NULL, SHGDN_FORPARSING,&Path);
    TV->Items->Clear();
    // Добавляем папку Desktop к средству просмотра дерева.
    AddNode(NULL, ishellfolder, StrRetToStr(NULL, &Name),
                         StrRetToStr(NULL,&Path), lpitemidlist);
}
//---------------------------------------------------------------
```

Содержимое листинга 30.45 не требует разъяснений, за исключением функции `AddNode()`, которая представлена в листинге 30.46.

Листинг 30.46. Функция `AddNode()`

```
void __fastcall TForm1::AddNode(TTreeNode *Node,
                  IShellFolder*ishellfolder, String FolderName,
                  String Path, LPITEMIDLIST pidl)
{
    TTreeNode *AddedNode;
    PMyRec MyRecPtr;
    MyRecPtr = new TMyRec;
    MyRecPtr->FirstExpand = true;
    MyRecPtr->ishlf = ishellfolder;
    AddedNode = TV->Items->AddChildObject(Node, FolderName, MyRecPtr);
    AddedNode->ImageIndex = GetImageIndex(FolderName, Path, pidl);
    AddedNode->SelectedIndex = GetImageIndex(FolderName, Path, pidl);
    if(AddedNode->Data)
        AddedNode = TV->Items->AddChild(AddedNode, "Dummy");
}
```

Функция `AddNode()` начинает свою работу с создания нового указателя на структуру `MyRec`. Эта структура предназначена для данных, которые будут сохраняться в каждом узле. Первый элемент структуры `FirstExpand` (типа `Boolean`) позволяет определить, расширялся ли ранее узел. Если это первое расширение узла, будут считаны объекты, входящие в состав этого узла (папки).

Второй элемент структуры `MyRec` имеет тип `IShellFolder`. Этот элемент предназначен для хранения объектов, считанных для узла. Структура `MyRec` определена в заголовочном файле `WinExplorer` (см. листинг 30.43).

Функция `AddNode()` также используется в обработчике события `OnExpanding` дерева, чтобы объекты узла считывались только при первом его расширении.

Обработчик события OnExpanding устанавливает свойство FirstExpands равным значению false. Если это свойство равно значению true, будет вызвана функция GetObject(). В противном случае происходит возврат без выполнения каких-либо действий. Функция GetObjects() показана в листинге 30.47.

Листинг 30.47. Функция GetObjects()

```
void __fastcall TForm1::GetObjects(TTreeNode *Node)
{
    TTreeNode*ANode;
    LPMALLOC lpMalloc;
    LPSHELLFOLDER lpshellfolder, lpshellsubfolder;
    LPENUMIDLIST lpenumidlist;
    LPITEMIDLIST lpitemidlist;
    String SubFolderName, SubFolderPath;
    STRRET Path, Name;
    ANode = Node->getFirstChild();
    ANode->Free();
    if (SUCCEEDED(SHGetMalloc(&lpMalloc)))
    {
        lpshellfolder = LPSHELLFOLDER(PMyRec(Node->Data)->ishlf);
        if (!lpshellfolder)return;
        if (SUCCEEDED(lpshellfolder->EnumObjects(Handle,
                        SHCONTF_FOLDERS | SHCONTF_NONFOLDERS |
                                          SHCONTF_INCLUDEHIDDEN,
                        &lpenumidlist)))
        {
            SHGetMalloc(&lpMalloc);
            while (lpenumidlist->Next(1, &lpitemidlist, NULL) == S_OK)
            {
                lpshellfolder->GetDisplayNameOf(lpitemidlist,
                                        SHGDN_NORMAL, &Name);
                lpshellfolder->GetDisplayNameOf(lpitemidlist,
                                      SHGDN_FORPARSING, &Path);
                SubFolderName = StrRetToStr(lpitemidlist, &Name);
                SubFolderPath = StrRetToStr(lpitemidlist, &Path);
                if(SUCCEEDED(lpshellfolder->BindToObject(
                            lpitemidlist, NULL, IID_IShellFolder,
                            (LPVOID *)&lpshellsubfolder)))
                    AddNode(Node, lpshellsubfolder,
                                SubFolderName, SubFolderPath,
                                lpitemidlist);
                else
                    AddNode(Node, NULL, SubFolderName,
                                SubFolderPath, lpitemidlist);
                lpMalloc->Free(lpitemidlist);
            } // конец цикла while
            lpenumidlist->Release();
        } //конец if
    } // конец if
}
```

Функция `GetObjects()` начинается с удаления пустого узла, который был вставлен функцией `AddNode()`. Затем она выделяет память с помощью функции `SHGetMalloc()`. Если выделение памяти прошло успешно, будет выполнена проверка, является ли текущий узел папкой. Если текущий узел — не папка, функция завершит работу без выполнения каких-либо действий.

Если текущий узел — папка, функция `GetObjects()` приступит к считыванию его объектов. Прежде всего с помощью метода `IShellFolder::EnumObject()` она получит указатель на интерфейс `IEnumIDList`. Затем, используя метод `Next()` интерфейса `IEnumIDList`, будут опрошены все объекты этой папки.

С помощью метода `IshellFolder::GetDisplayNameOf()` эта функция прочитает имя и путь, а также получит интерфейс `IshellFolder` каждого объекта в папке. Затем вся эта информация будет присоединена к каждому объекту.

После считывания всей информации о каждом объекте в папке, выделенная память освобождается с помощью метода `Free()` интерфейса `IMALLOC`.

Работа этой программы показана на рис. 30.17.

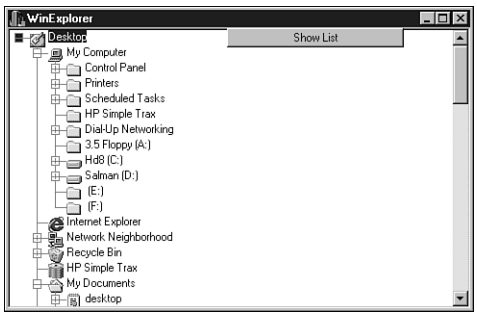


Рис. 30.17. Программа WinExplorer в действии

Резюме по вариации на тему Windows Explorer

WinExplorer — довольно простая программа, демонстрирующая возможность использования пространства имен оболочки. Чтобы создать приложение, действительно выполняющее функции Windows Explorer, необходимо проделать большой объем дополнительной работы. Тем не менее WinExplorer может быть неплохим стартом на пути в этом направлении.

Работа с NT-службами (NT Services)

В поисках примеров создания NT-служб мне попалась лишь одна статья "Writing NT Services," Part 1, написанная Кентом Рейсдорфом (Kent Reisdorph). Это прекрасная статья. Часть I называется "Understanding NT Services" (Понятие о NT-службах) и включает описание механизма, заставляющего работать эти программы. К сожалению, приведенная в качестве примера программа может лишь раз в секунду издавать звуковой сигнал, оповещая тем самым о своей работе, поэтому нам следует подыскать что-нибудь более полезное. Часть II статьи Кента уже доступна для желающих и более содержательна. Ее можно найти по адресу: `http://www.cbuildermag.com`.

А пока давайте-ка разберемся, в чем суть задачи. Компания 3M Corporation заработала миллиарды на создании таких новомодных продуктов, как PostIt Note (служба мгновенных

сообщений). С этого момента мы будем называть их "stickums" (от *Stickums Service*), или мгновенными сообщениями.

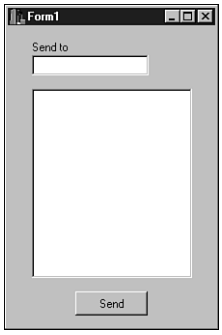
Идея же состоит в следующем.

1. Пользователь A желает сообщить что-нибудь пользователю B.
2. Некоторая служба (которую мы создадим) находит адрес пользователя B и отправляет ему сообщение.
3. Пользователь B видит сообщение (stickum) на своем экране.

Для реализации описанного алгоритма мы создадим три отдельные программы, именуемые Stickums, Stickem и SendMsg. Одна программа будет инсталлирована как NT-служба (Stickums). Она будет "слушать" и вызывать другую программу (Stickem), которая реально отобразит сообщение на нужном компьютере. Обе эти программы инсталлируются только в среде Windows NT. Наконец, у нас есть программа-клиент (SendMsg), которая позволяет составлять сообщения для отправки.

Программа SendMsg

Рассмотрим программу SendMsg, "лицо" которой показано на рис. 30.18. Эта форма вряд ли получит приз за лучший образец GUI-программирования, она, скорее, может претендовать на "звание" самой неказистой формы всех времен и народов. Однако у нее есть одно неоспоримое преимущество — она чрезвычайно проста.



В строке редактирования под надписью Send To (Кому) пользователь A вводит имя системы, в которую он хотел бы послать мгновенное сообщение (stickum). В большинстве компаний есть компьютеры с такими именами, как `Bob` (Боб) или `Support` (Поддержка), но не исключены и другие, более замысловатые, например `1ub06a`. Безусловно, если вы захотите узнать о работе в сети конкретного пользователя, вам предстоит "хождение по сетевым мукам". Подключен ли к сети `Bob`? Гм-м-м. Дайте подумать, это не пользователь ли `5xyy7a2b`?

А пока продолжим. В центре формы находится Memo-поле, предназначенное для ввода вашего сообщения, а в нижней ее части — кнопка Send (Отправить), по щелчку на которой и должно все "закрутиться".

Рис. 30.18. Главная форма программы SendMsg

А теперь сразу "быка — за рога", т.е. рассмотрим текст программы, которую можно найти на прилагаемом к книге компакт-диске.

Программа не содержит никакого кода инициализации. Первое, что заслуживает внимания, — процедура `SendButtonClick()`.

```
void __fastcall TForm1::SendButtonClick(TObject *Sender)
{
    DWORD *dwIPAddr;
    DWORD IPAddr;
    char work [80];
}
```

Обратите внимание на закомментированную структуру в следующем фрагменте программы. Она необязательна, но поможет понять суть происходящего.

```
/*
struct hostent
```

```
{
    char FAR            *h_name;
    char FAR *FAR *h_aliases;
    short               h_addrtype;
    short               h_length;
    char FAR *FAR *h_addr_list;
}
*/
```

Теперь нам нужно получить IP-адрес компьютера-приемника. Это делается с помощью кода, представленного в листинге 30.48.

Листинг 30.48. Сетевой поиск компьютера по имени

```
hostent *pHost =gethostbyname(Edit1->Text.c_str());
if (pHost != NULL)
{
    dwIPAddr = (DWORD *)(*pHost->h_addr_list);
    IPAddr = *dwIPAddr;
     sprintf(work,"%d.%d.%d.%d",
                                                    LOBYTE(LOWORD(IPAddr )),
                                                    HIBYTE(LOWORD(IPAddr )),
                                                    LOBYTE(HIWORD(IPAddr )),
                                                    HIBYTE(HIWORD(IPAddr )));
     ClientSocket1->Address = String(work);
```

Эта форма содержит компонент TClientSocket. Все, что мы хотим здесь сделать, — открыть его. Если у нас возникнут какие-либо проблемы, мы сообщим о них пользователю. Этот код показан в листинге 30.49.

Листинг 30.49. Открытие сокета клиента (Client Socket)

```
     try
     {
         ClientSocket1->Open();
     }
     catch(...)
     {
         AnsiString Problem = "Error opening: " + String(work)+ " Port 711";
          Application->MessageBox(Problem.c_str(), "Sorry", MB_OK);
     }
}
else
     Application->MessageBox("Couldn 't find recipient", "Error ", MB_OK);
}
```

Следующая процедура выполняет самую большую часть работы. Она вызывается, когда компонент TClientSocket в нашей службе передачи сообщений соединяется с компонентом TServerSocket.

```
void __fastcall TForm1::ClientSocket1Write(TObject *Sender,
                                           TCustomWinSocket *Socket)
```

```
{
    char host [40];
    gethostname(host, 40);
    AnsiString Buffer = "\" From: " + String(host)+ "\" ";
     for (int i = 0; i < Memo1->Lines->Count; i++)
          Buffer += " \"" ++Memo1->Lines->Strings [i ] + "\" ";
     Socket->SendText(Buffer);
     Close();
}
```

Как видите, мы получаем локальное имя (hostname) и предваряем им сообщение, которое пользователь ввел в Memo-поле. Эта процедура вызывается, когда сокет соединяется с нашей службой, на которую он и был ориентирован. Нет соединения — нет и выполнения. Закроем форму `SendMsg` и рассмотрим саму службу мгновенных сообщений.

Служба мгновенных сообщений (Stickums Service)

Самое время создать службу, которая будет "слушать" сообщения. Из меню File⇨New выберите команду Service Application (Служебное приложение).

Тем самым будет создана оболочка с именем `Service1`. Сохраните ее в файле `Stickums.bpr` в отдельной папке. Форма для службы сообщений не является формой в общепринятом смысле, по крайней мере она не используется для того, чтобы вы могли видеть службу. Смысл ее существования в том, чтобы в нее можно было добавлять компоненты. Этим мы и займемся, т.е. поместим в нее из вкладки Internet палитры компонентов сокет сервера (server socket).

Затем отредактируем свойства сокета сервера.

Оба свойства `AllowPause` и `AllowStop` могут иметь значения `true`. Мы не хотим иметь никаких зависимостей. Свойству `DisplayName` присвоим значение `stickums`, а свойству `ErrorSeverity` — значение `esNormal`. На самом деле нас больше всего интересуют только два свойства. Одно из них — свойство `ServiceType` с возможными значениями `stDevice`, `stFileSystem` или `stWin32`. Поскольку мы создаем не драйвер устройства и не файловую систему, то выбираем для этого свойства значение `stWin32`, которое означает службу с использованием общего потока (shared thread service).

Вторым свойством, представляющим для нас особый интерес, является свойство `StartServiceName`. Название не совсем соответствует его назначению. На самом деле это имя учетной записи, под которой мы собираемся работать. Специальное значение `.\LocalSystem` определяет службу как интерактивную (interactive service), но не интерактивную в смысле вашей возможности что-то ей сообщать, а интерактивную в том смысле, что она будет действовать на текущем рабочем столе. Неинтерактивные службы работают под учетной записью действующего пользователя и наследуют те привилегии, которые дает им эта учетная запись. В соответствии с документацией, программный код, который мы будем использовать, не должен работать с неинтерактивными службами. При определенных усилиях его можно заставить работать, но намного проще (и менее болезненно) использовать метод `LocalSystem`.

Итак, установим свойство `StartType` равным значению `stManua` (или можно было бы установить его равным значению `stAutomatic`). Теперь рассмотрим реальный код нашей службы сообщений, представленный в листинге 30.50. Этот код генерируется автоматически. В нем предусмотрены операции инициализации и связи с менеджером служб (Service Control Manager).

Листинг 30.50. Программа службы сообщений Stickums

```
#include "Unit1.h"
//---------------------------------------------------------------
#pragma package(smart_init)
#pragma resource "*.dfm"
TStickum *Stickum;
//---------------------------------------------------------------
__fastcall TStickum::TStickum(TComponent*Owner): TService(Owner)
{
}
//---------------------------------------------------------------
TServiceController __fastcall TStickum::GetServiceController(void)
{
     return (TServiceController) ServiceController;
}
//---------------------------------------------------------------
void __stdcall ServiceController(unsigned CtrlCode)
{
     Stickum->Controller(CtrlCode);
}
```

Теперь нужно побеспокоиться о событии OnExecute. Для этого откройте менеджер свойств формы, выберите список событий и дважды щелкните на событии OnExecute. В его обработчик введите следующий код.

```
void __fastcall TStickum::StickumExecute(TService *Sender)
{
    while (!Terminated)
    {
        ServiceThread->ProcessRequests(false);
    }
}
```

Здесь следует дать один комментарий. Нам нужно, чтобы компонент TServiceApplication обрабатывал сообщения до тех пор, пока соответствующая служба не завершит свою работу. Это дает возможность выполниться всем службам. Нам также нужно воспользоваться еще двумя событиями: OnStart и OnStop. Код их обработки приведен в листинге 30.51.

Листинг 30.51. Активизация сокета сервера

```
void __fastcall TStickum::StickumStart(TService *Sender, bool &Started)
{
    ServerSocket1->Active = true;
    Started = true;
}
//---------------------------------------------------------------
void __fastcall TStickum::StickumStop(TService *Sender, bool &Stopped)
{
    ServerSocket1->Active = false;
    Stopped = true;
}
```

Прежде всего необходимо активизировать наш сокет сервера. Обратите внимание, что мы также уведомляем источник вызова о начале своей работы. Процедура останова действует прямо противоположным образом. На данном этапе служба сообщений должна просто "сидеть в засаде", ожидая, пока кто-нибудь не подключится. Когда это произойдет, служба получит уведомление (в виде события) со стороны другого сокета (сокета клиента), и сокету сервера будет предложено прочитать посланное сообщение. Реализация описанной процедуры представлена в листинге 30.52.

Листинг 30.52. Программа службы сообщений, обеспечивающая "выход" на рабочий стол

```
void __fastcall TStickum::ServerSocket1ClientRead(
                     TObject *Sender, TCustomWinSocket *Socket)
{
     AnsiString Buffer = "";
     AnsiString Cmd = "Stickem.exe ";
     Buffer = Socket->ReceiveText();
     Cmd += Buffer;

     STARTUPINFO si;
     PROCESS_INFORMATION pi;
     si.cb = sizeof(STARTUPINFO);
     si.lpReserved = NULL;
     si.lpTitle = NULL;
     si.lpDesktop = "WinSta0\\Default";
     si.dwX = si.dwY = si.dwXSize = si.dwYSize = 0L;
     si.dwFlags = 0;
     si.wShowWindow = SW_SHOW;
     si.lpReserved2 = NULL;
     si.cbReserved2 - 0;
     if (CreateProcess(NULL, Cmd.c_str(), NULL, NULL, FALSE, 0,
                                                  NULL, NULL, &si, &pi))
     {
          CloseHandle(pi.hProcess);
          CloseHandle(pi.hThread);
     }
}
```

Программа начинается с получения от подключившегося сокета некоторого текста, который присоединяется в конец команды. Затем создается структура `STARTUPINFO`, после чего вызывается API-функция `CreateProcess()`. Если работа этой функции прошла успешно, мы закрываем дескрипторы для данного процесса и основного потока. Они нам больше не нужны, и если бы мы этого не сделали, новый процесс оставался бы в системе неограниченно долго.

Рассмотрим инициализацию переменной `si.lpDesktop`, члена структуры `STARTUPINFO`. Здесь есть одна хитрость. Наша служба сообщений работает на "ничейной территории". Она имеет свой виртуальный компьютер и рабочий стол, который не отображается на экране. В действительности, мы сообщаем API-функции, что хотели бы создать новый процесс под управлением виртуального компьютера текущего пользователя, используя при этом его рабочий стол. Реализация этого намерения в виде жесткого кодирования (с помощью оператора присвоения) — не лучший способ решения этой проблемы. Компания Microsoft заявила, что в будущем пользователь сможет иметь более одного рабочего стола!

Ну, а пока мы имеем то, что имеем, т.е. не слишком лаконичную и непростую для понимания программу. Поэтому просто не забывайте, что, пока обещанное будущее не наступило, мы плывем под “пиратским флагом”.

Мы еще не отредактировали свойства сокета сервера. Для этого нужно дважды щелкнуть на событии `OnClientRead` и ввести код, представленный в листинге 30.52.

Теперь перейдем во вкладку Properties (Свойства) и установим соответствующий порт.

Номер порта 711 звучит хорошо, не правда ли? Должно быть, он не зарезервирован для других служб. Программы, предназначенные для реализации протоколов FTP и SNMP зарезервировали определенные сокеты или порты. Они хранятся в системной папке `SERVICES`. Если вас это интересует, можете туда заглянуть. Выбранный вами порт необходимо установить здесь и в программе SendMsg. Если вы хотите делать это в динамическом режиме, добавьте еще одну запись в файл `SERVICES` и вместо порта заполните Service type. Теперь нужно скомпилировать сервер. Сделайте это сейчас, и можно будет приступать к этапу тестирования.

После “безошибочной” компиляции и компоновки необходимо инсталлировать службу сообщений. Для этого переключитесь в режим выполнения команд DOS и перейдите в папку, содержащую вашу программу. Созданная нами служба сообщений уже достаточно интеллектуальна (благодаря Borland), чтобы инсталлировать и деинсталлировать саму себя (безусловно, при наличии соответствующего параметра командной строки). На рис. 30.19 показана командная строка инсталляции службы сообщений. Она будет вступать в контакт с менеджером служб Service Control Manager и передавать запрос. Не забывайте, что она работает только в среде Windows NT.

Предположим, что команда выполнилась нормально. В этом случае менеджер служб должен отреагировать диалоговым окном, подтверждающим, что служба приступила к своим обязанностям.

Итак, служба сообщений инсталлирована, но работать не начала. Чтобы “подтолкнуть” ее, выберите команду Windows Start⇨Settings⇨Control Panel (Пуск⇨Настройка⇨Панель управления). Откроется окно панели управления. Найдите элемент Services и дважды щелкните на нем.

Обратите внимание, что программа Stickums инсталлирована, но мы запросили ручной старт. Дело в том, что в случае наличия каких-либо ошибок вряд ли вам будет приятна их демонстрация при каждой начальной загрузке Windows. Пока просто щелкните на кнопке Start. Откроется маленькое окно состояния, сообщающее о том, что Windows пытается запустить программу услуг, а затем финальное сообщение о благополучном исходе или возникших проблемах. В последнем случае вернитесь на несколько страниц назад и повторите процедуру. Если у вас вариант успешного завершения, можете продолжить чтение.

Рис. 30.19. Инсталляция службы сообщений из командной строки

Программа клиента Stickem

До сих пор мы рассматривали GUI-интерфейс пользователя и службу сообщений, которая будет ожидать подключения, а затем (когда оно произойдет) вызовет другую программу и передаст ей в виде сообщения параметры командной строки. Теперь мы познакомимся с вызываемой программой, которая реально отображает сообщение для получателя.

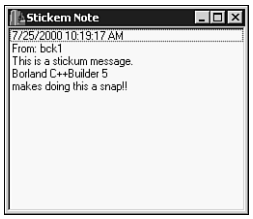


Рис. 30.20. Программа Stickem в действии

Она называется Stickem и показана на рис. 30.20. Подобно PostIt Note, она выглядит очень нарядно, чтобы привлечь ваше внимание яркими цветами. Как и PostIt Note, она не отводит много места для отображения текста. Но, в отличие от PostIt Note, если текстовая строка слишком длинна или имеет слишком много строк, наша программа предложит полосы прокрутки.

Вы заметите, что перед именем компьютера отправителя добавлены значения даты и времени. С моей точки зрения, было бы лучше, если бы у компьютера было имя `Pete`; тогда бы получатель знал, от кого пришло сообщение. Полная программа Stickem состоит из трех строк "жестко написанного" кода, которые выполняются в процедуре инициализации.

```
ListBox1->Items->Add(Now().DateTimeString());
for (int i = 1; i <= ParamCount(); i++)
    ListBox1->Items->Add(ParamStr(i));
```

Первая строка вставляет строку даты и времени. Следующие две строки заполняют окно списка информацией, отправленной службой сообщений. Среда C++Builder (по сравнению со стандартом языка C) значительно облегчает жизнь программиста: не нужно беспокоиться о создании окна списка, о его размещении на экране и передаче в него данных. Конечно, RAD-разработка не лишена недостатков, но неоспоримо то, что C++Builder — прекрасная реализация языка C++, ведь все, что можно сделать в среде Visual C++ (VC++), можно сделать и в среде C++Builder.

Резюме по службе мгновенных сообщений

Если вы дочитали до этого места, то, возможно, у вас уже работает служба, которая может выводить сообщения (stickums) на рабочие столы ваших сотрудников. При ее инсталляции не забудьте поместить файл `Stickem.exe` в системную папку Windows. Желаем удачи и получения приятных сообщений!

Использование криптографии

Как известно, секреты не могут долго оставаться секретами. Не сочтите это укором для тех, кто обожает посплетничать. Просто это отражение реального состояния мира. Не только правительство Соединенных Штатов может прослушивать ваши коммуникационные линии. Прослушивающих станций слишком много по всему миру. Телефону больше нельзя доверять никаких тайн, равно как и передаче информации через Internet. Все, что вы скажете, может и будет использовано против вас. Вы думаете, вас спасет то, что никто не знает вашего языка? Поинтересуйтесь патентом № 4775956, озаглавленным в США "Method and System for Information Storing and Retrieval Using Word Stems and Derivative Pattern Codes Representing Families of Affixes" (Метод и система хранения и восстановления информации с помощью основ слов и кодов производных образцов, представляющих семейства аффиксов). Все еще неубедительно? Тогда про-

читайте статью IEEE “N-gram Statistics for Natural Language Understanding and Text Processing” (Статистические данные по проблеме понимания естественного языка и обработке текстов), написанную К. Суеном (C. Suen), Vol. PAMI-1, No. 2, April 1979.

Эти патенты и процедуры используются Агентством национальной безопасности США для контроля коммуникаций по всему миру. Суть патента состоит в том, что существуют программы, которые в поиске смысла реагируют не на слова, а на последовательность фонем. Короче говоря, этот принцип работает для любого языка, а не только для английского. Для получения дополнительных сведений на эту тему попробуйте найти в Web информацию, задав в качестве искомого элемента словосочетание “Project Echelon”, или заглянуть в Web-узлы, например, по адресу: `http://politicaltexan.com/wwwboard/messages/58.html`.

И конечно же, пускать все на самотек — себе во вред. Для поддержки конфиденциальности необходимо иметь способ надежной шифровки текста, а ваши получатели должны уметь расшифровывать его. В этом разделе мы рассмотрим:

- приложение, создающее “одноразовый блокнот” (*o*ne *t*ime *p*ad — OTP) на основе одного из самых известных шифровальных алгоритмов;
- приложение кодирования текста с помощью нашего OTP-алгоритма;
- приложение, предназначенное для декодирования зашифрованного текста.

Решение

Этому решению мы обязаны леди Аде Байрон (Lady Ada Byron), графине Ловелас (Lovelace). Многие считают, что она была первой из тех, кто применил кодовые подстановки для общения. Алгоритм одноразового блокнота (OTP) был впервые описан Мэйером Моуборном (Major Mauborne) и Джилбертом Вернамом (Gilbert Vernam) из компании Bell Labs в 1917 году. OTP-алгоритму не присущ дефект большинства систем подстановки, в которых конкретная буква или слово всегда соответствует одному и тому же символу. И при его надлежащем использовании не возникает проблем, присущих таким классическим алгоритмам шифрования, как RSA, DES или IDEA. Эти системы делятся на три категории. В первую входят сложные алгоритмы, использующие перестановки, которые повторяются несколько раз. Особенность второй категории — зависимость от простых чисел, причем слишком больших для обычных вычислительных аппаратных средств. По сравнению с ними, алгоритм OTP, использующий тривиальную схему подстановки, — сама простота. Те, кто хочет больше узнать о таких алгоритмах шифрования, как OTP, RSA, DES и IDEA, могут обратиться по адресу: `http://www.cs.bris.ac.uk/Teaching/Resources/COMS71301/Cryptography/crypto.html`.

Однако существует несколько правил для по-настоящему надежного кодирования.

- Ключ, который используется для шифрования/дешифрования сообщения, должен быть сгенерирован с использованием настоящего случайного процесса — наподобие радиоактивного распада, шума в линии электропередачи или помех от плохо настроенного радиоприемника, преобразованных в числовое значение.
- Как отправитель, так и получатель должны иметь идентичную копию ключа. Это может быть неудобно.
- OTP-алгоритм должен примяться только один раз. Повторное использование влияет на ключ, чувствительный к дешифрированию. Это делает OTP-алгоритм неприемлемым для крупномасштабных сред. После Второй мировой войны западные разведывательные службы взломали русский OTP-код именно по причине его многоразового использования. Питер Райт (Peter Wright) описывает это как операцию VENONA в книге *Spycatcher*.

В этом разделе мы рассмотрим пример OTP-системы, которая была реализована в три этапа в виде трех приложений. Их можно найти в папке `Crypto` на прилагаемом к книге компакт-диске. Мы не будем создавать законченное приложение, рассмотрим лишь методы криптографии, используемые в этих программах.

Первый этап заключается в генерации случайного ряда чисел, который будет использоваться в качестве *ленты одноразового использования*. Речь идет о приложении MakePad (`MakePad\MakePad.bpr`). Второй этап состоит в использовании ленты, сгенерированной приложением MakePad для шифрования текстовых файлов. Имеется в виду приложение Encode (`Encode\Encode.bpr`). Третий и последний этап представляет собой дешифрирование текстовых файлов, зашифрованных посредством приложения Encode, с помощью ленты, сгенерированной приложением MakePad. Заключительный этап реализован в приложении Decode (`Decode\Decode.bpr`).

Сначала рассмотрим процесс генерации ленты одноразового использования, используя для этого проект `MakePad.bpr`. Текст программы генерации OTP-кода приведен в листинге 30.53.

Листинг 30.53. Генерация ленты одноразового использования

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

#include "PadForm.h"
#include <dir.h>
#include <stdlib.h>

//---------------------------------------------------------------
#pragma package(smart_init)
#pragma resource "*.dfm"
TForm1 *Form1;
#define padsize 131072
char Pad [padsize];
char szFileName [MAXFILE+4];
int iFileHandle;
int iLength;

//---------------------------------------------------------------
__fastcall TForm1::TForm1(TComponent*Owner)
     : TForm(Owner)
{
     randomize();
     memset (Pad, 0, padsize);
     for (int i = 0; i < padsize; i++)
     {
          Pad [i] == (char) random(256);
          Application->ProcessMessages();
          Pad [i] == Pad [i]^(char) random(256);
     }

     if (SaveDialog1->Execute() == IDOK)
     {
          try
```

```
        {
             if (FileExists(SaveDialog1->FileName))
             {
                  fnsplit(SaveDialog1->FileName.c_str(), 0, 0, szFileName, 0);
                  strcat(szFileName, ".BAK");
                  RenameFile(SaveDialog1->FileName, szFileName);
             }

             iFileHandle = FileCreate(SaveDialog1->FileName);
             iLength = padsize;
             FileWrite(iFileHandle,(char*)&iLength, sizeof(iLength));
             FileWrite(iFileHandle, Pad, padsize);
             FileClose(iFileHandle);
        }
        catch(...)
        {
             Application->MessageBox("Sorry, can 't create the pad",
                                     "File Error", IDOK);
        }
    }
    Application->Terminate();
}
```

Прежде всего обратите внимание на то, что размер создаваемой ленты определяется числом 131 072. Это значение вы можете изменить. Однако имейте в виду, что оно должно быть достаточно большим (чем больше, тем лучше).

При создании формы выполняется процесс рандомизации, который обязывает компилятор сгенерировать начальное число для последовательности случайных чисел на основе системного времени или других переменных вашего компьютера. Вы также можете задать его сами, если хотите, чтобы генератор случайных чисел каждый раз воспроизводил одну и ту же последовательность. Если вас еще не клонит ко сну, то вы могли бы заметить, что генератор случайных чисел вовсе не является настоящим генератором случайных чисел. Как упоминалось выше, OTP-метод работает корректно только в том случае, если он основан на действительно случайной последовательности. А здесь мы имеем дело с псевдослучайной последовательностью. Обратите внимание, что после рандомизации мы вводим в то, что должно быть лентой при шифровании, случайные символы, а затем выполняем операцию исключающего ИЛИ (XOR) между ними и следующим случайным символом. Причем до выполнения операции исключающего ИЛИ мы вызываем метод класса `TApplication` для обработки любых системных сообщений. Конечно, это не создаст ленты с идеальным случайным распределением, но избавит вас от необходимости следить за распадом атомов или слушать радиопомехи. (Последний вариант, кстати, может оказаться неплохой идеей, если, конечно, добавить микрофон и звуковую карту.)

Остальная часть программы просто запрашивает имя файла для OTP-метода и резервирует любой существующий файл с этим именем, после чего OTP-вариант записывается на диск. Форма, связанная с этой программой, никогда не “видит дневного света”, поскольку в конце процедуры инициализации мы заставляем форму самоликвидироваться.

Полученная в результате лента (если открыть ее с помощью текстового редактора) будет выглядеть примерно так, как показано на рис. 30.21.

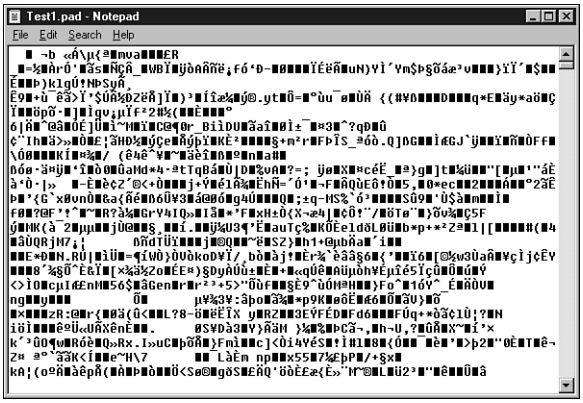


Рис. 30.21. Пример одноразового блокнота, отображенного с помощью редактора Notepad

Большинство символов, показанных на рис. 30.21, не имеет другого экранного представления, кроме зачерненного квадрата или пробела. И так будет выглядеть наш зашифрованный файл? Нет. OTP-вариант, показанный на рис. 30.21, сгенерирован с использованием всех возможных значений между 0 и 255, чтобы можно было шифровать двоичные файлы. Это не самый удачный формат для результата шифрования, поскольку его нельзя достаточно успешно передать через все системы. Поэтому мы создадим простой читабельный ASCII-файл.

Шифрование файлов

Конечно, интерфейс пользователя, показанный на рис. 30.22, далек от совершенства. В нем не предусмотрены никакие справочные файлы или меню — два элемента, которые обязательно должны присутствовать в законченном продукте. Однако у него есть преимущество: простота.

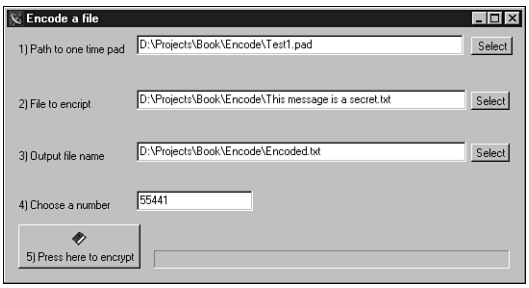


Рис. 30.22. Программа Encode в действии

Для шифрования файла пользователь должен выполнить пять простых действий.

1. Выберите одноразовую ленту.
2. Выберите файл, подлежащий шифрованию.
3. Выберите файл результата.

4. Выберите произвольное число. Это число будет начальным смещением в одноразовой ленте, позволяющее еще более усложнить процесс расшифровки для тех, кто занимается этим самовольно.
5. Щелкните на кнопке, чтобы начать процесс шифрования.

Теперь рассмотрим каждую из функций. Выбор одноразовой ленты выполняется следующим образом.

```
void __fastcall TForm1::SBOpenPadClick(TObject *Sender)
{
    if (OpenPadDialog->Execute() == IDOK)
    {
        Edit1->Text = OpenPadDialog->FileName;
        GotPad = true;
    }
}
```

Для выбора файла используется компонент `OpenDialog`. В текстовое поле `TEdit` помещается имя файла и устанавливается значение переменной `GotPad` булева типа, которое проверяется позже.

Следующие два действия почти идентичны этому и представлены в листинге 30.54.

Листинг 30.54. Запрос у пользователя имени файла

```
void __fastcall TForm1::SBOpenFileToEncryptClick (TObject *Sender)
{
    if (OpenFileToEncryptDialog->Execute() == IDOK)
    {
        Edit2->Text = OpenFileToEncryptDialog->FileName;
        GotInput = true;
    }
}
//---------------------------------------------------------------
void __fastcall TForm1::OpenOutputFileClick (TObject *Sender)
{
    if (SaveOutputDialog->Execute() == IDOK)
    {
        Edit3->Text = SaveOutputDialog->FileName;
        GotOutput = true;
    }
}
```

Когда пользователь щелкает на последней кнопке, мы должны убедиться в том, что пользователь идентифицировал все необходимые файлы, как показано в листинге 30.55.

Листинг 30.55. Как убедиться в том, что пользователь предоставил имена всех нужных файлов

```
void __fastcall TForm1::EncryptButtonClick (TObject *Sender)
{
    char Output [80];
    AnsiString Offset;
    int off;
```

```
    int disp;

    if (!(GotOutput))
    {
          Application->MessageBox("Specify output", "User Error", IDOK);
          return;
     }
     if (!(GotInput))
     {
          Application->MessageBox("Specify input", "User Error", IDOK);
          return;
     }
     if (!(GotPad))
     {
          Application->MessageBox("Specify one time pad","User Error", IDOK);
          return;
     }
     if (!FileExists(Edit1->Text))
     {
          Application->MessageBox("Cant find one time pad", "File Error", IDOK);
          return;
    }
    if (!FileExists(Edit2->Text))
    {
          Application->MessageBox("Cant find user input", "File Error", IDOK);
          return;
    }
    if (Edit3->Text == Edit2->Text)
    {
        Application->MessageBox("Input and output names can 't be the same",
                                "User Error", IDOK);
        return;
    }
}
```

Следующий фрагмент программы несколько иного плана. Он разбивает результирующее имя файла на составные части и присоединяет к имени расширение `.BAK`. Если `.BAK`-файл уже существует, то он удаляется. Существующий файл с результирующим именем затем переименовывается с использованием расширения `.BAK`.

```
if (FileExists(Edit3->Text))
{
    fnsplit(Edit3->Text.c_str(), 0, 0, szFileName, 0);
    strcat(szFileName, ".BAK");
     if (FileExists(String(szFileName)))
          DeleteFile(String(szFileName));
     RenameFile(Edit3->Text, szFileName);
}
```

Затем создается результирующий файл.

```
oFileHandle =FileCreate(Edit3->Text);
```

Представленное пользователем смещение изменятся таким образом, чтобы оно было меньше размера одноразовой ленты. Это число в строковом виде первым записывается в файл результата. Получателю не нужно знать это число.

```
off = (StrToInt(Edit4->Text)) % padsize;
Offset = String(off);
sprintf(Output, "%s ", Offset.c_str());
FileWrite(oFileHandle, Output, strlen(Output));
```

В обработке файла стоит использовать try/catch-логику. Курсор при этом устанавливаем в виде песочных часов, а затем читаем в память одноразовую ленту.

```
try
{
    Cursor = crHourGlass;
    iFileHandle = FileOpen(Edit1->Text, fmOpenRead);
    FileRead(iFileHandle, (char *) &iFileLength, sizeof(iFileLength));
    FileRead(iFileHandle, Pad, iFileLength);
    FileClose(iFileHandle);
```

Длина файла, подлежащего шифрованию, неизвестна, поэтому перед выделением памяти мы ищем конец файла, чтобы определить его длину. После этого файл считывается в выделенную для него область.

```
    i2FileHandle = FileOpen(Edit2->Text, fmOpenRead);
    i2FileLength = FileSeek(i2FileHandle, 0, 2);
    FileSeek(i2FileHandle, 0, 0);

    pszBuffer = new char[i2FileLength+1];
    FileRead(i2FileHandle, pszBuffer, i2FileLength);
    FileClose(i2FileHandle);
```

Все шифрование выполняется в следующем цикле. Индекс `i` наращивается до размера входного файла. Переменная отклонения `disp` инкрементируется до тех пор, пока текущий символ не будет равен символу в одноразовой ленте. Если накопленное значение переменной `disp`, сложенное со значением смещения (`off`), станет больше длины одноразовой ленты, смещение устанавливается равным нулю. Обратите внимание, что переменная `disp` не сбрасывается в нуль до следующего входа в цикл.

```
for (int i=0; i < i2FileLength; i++)
{
    disp = 0;
    while (pszBuffer[i] != Pad[off+disp])
    {
        disp++;
        if ((off +disp) > iFileLength)
            off = 0;
    }
```

После обнаружения символа значение переменной `disp` форматируется и записывается в выходной файл. При этом (обратите внимание!) сам символ в выходной файл никогда не записывается.

```
sprintf(Output, "%i ", disp);
FileWrite(oFileHandle, Output, strlen(Output));
```

После записи значения переменной disp в результирующий файл смещение увеличивается на величину этого значения и еще на единицу, а затем обновляется индикатор выполнения шифрования. Мы также не забыли об обработке сообщений, которые могут ожидать "своей участи". Это стоит делать особенно в тех случаях, когда программа работает в циклах.

```
off += disp +1;
ProgressBar1->Position = ((float)i * 100.0) / (float)i2FileLength;
ProgressBar1->Update();
Update();
Application->ProcessMessages();
```

После завершения описанной работы можно освободить выделенную ранее память и закрыть файл результата.

```
delete [] pszBuffer;;
FileClose(oFileHandle);
ProgressBar1->Position = 0;
ProgressBar1->Update();
```

Возможные при выполнении файловых операций ошибки обрабатываются catch-логикой, а пользователю при этом отправляется сообщение об ошибке.

```
catch(...)
{
    Application->MessageBox("Errors encoding.", "Program Error", IDOK);
}
Cursor = crDefault;
```

И наконец, мы возвращается к стандартному изображению курсора, чтобы пользователь знал о завершении процесса.

Безусловно, эта программа по сложности несравнима с задачами, решаемыми в области ракетостроения. Мы просто автоматизировали подстановку смещения для символа. Очевидно, что только отправитель и получатель должны иметь копию шифровальной ленты. Поскольку лента не содержит скрытой информации о зашифрованном сообщении, отклонения для определенного символа не содержат сведений об этом символе, если, конечно, не иметь копии шифровальной ленты.

Используя в качестве входного материала текст, показанный на рис. 30.23, и сгенерированный выше блокнот, мы получаем результат, который имеет вид файла, показанного на рис. 30.24.

Рис. 30.23. Сообщение, подлежащее шифрованию

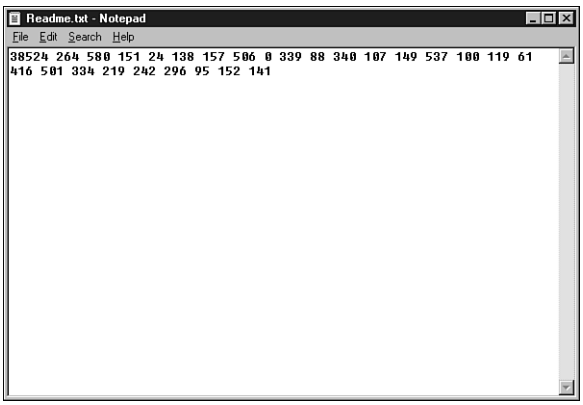


Рис. 30.24. Зашифрованное сообщение

Без копии одноразовой ленты, используемой при шифровании текста, его расшифровка невозможна. Этот метод имеет один существенный недостаток. Размер выходного файла (файл результата) будет в несколько раз больше входного. С другой стороны, с помощью этого метода можно зашифровать любой файл, а не только текстовый. Кроме того, зашифрованные файлы имеют форму, которая легко передается по средствам коммуникации. А если проблемой является размер, то вспомните о возможности упаковывать файлы.

Дешифрирование файла

Сходство между программами шифрования и дешифрования очевидно. Единственное визуальное отличие в том, что на экране дешифрования (показанном на рис. 30.25) пользователю не предлагается ввести “случайное” смещение. Используемое смещение будет прочитано из зашифрованного (в данном случае входного) файла.

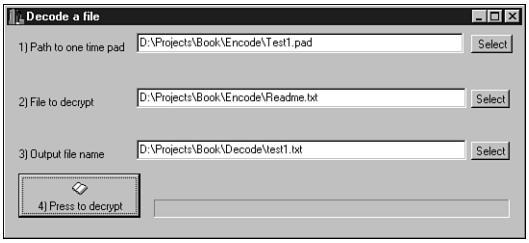


Рис. 30.25. Программа Decode в действии

Эти две программы очень близки по логике, поэтому было бы излишне тратить время читателя на рассмотрение уже знакомого кода. Однако стоит взглянуть на представленный в листинге 30.56 фрагмент программы, в котором дешифрируются реальные данные. Начнем с открытия соответствующих файлов и установки экранного курсора.

Листинг 30.56. Чтение содержимого OTP-ленты и зашифрованного файла

```
try
{
    Cursor = crHourGlass;
    iFileHandle = FileOpen(OpenDialog1->FileName, fmOpenRead);
```

```
FileRead(iFileHandle, (char *)&iFileLength, sizeof(iFileLength));
FileRead(iFileHandle, Pad, iFileLength);
FileClose(iFileHandle);

i2FileHandle = FileOpen(OpenDialog2->FileName, fmOpenRead);
i2FileLength = FileSeek(i2FileHandle, 0, 2);
FileSeek(i2FileHandle, 0, 0);

pszBuffer = new char[i2FileLength+1];
FileRead(i2FileHandle, pszBuffer, i2FileLength);
FileClose(i2FileHandle);
```

Символьный указатель используется для указания на данные, а предел устанавливается в конце буфера.

```
p = pszBuffer;
end = pszBuffer;
end += i2FileLength;
```

Функция `Strchr()` просматривает строку в попытке найти конкретный символ. В данном случае мы выполняем поиск первого пробела и преобразуем его в двоичный нуль. Затем мы преобразуем первое обнаруженное в файле число в смещение одноразовой ленты и восстанавливаем указатель на нужный символ. Этот процесс показан в листинге 30.57.

Листинг 30.57. Дешифрирование зашифрованного файла

```
p2 = strchr(p,' ');
*p2 = '\0';
off = atol(p);
p = p2 +1;

while (p < end)
{
     p2 = strchr(p, ' ');
     *p2 = '\0';
     disp = atol(p);
     p = p2 +1;

     if ((off + disp) > iFileLength)
          off = 0;

     FileWrite(oFileHandle, (char *)&Pad[off+disp], 1);
     off += disp + 1;
     ProgressBar1->Position = ((1.0 -
      (((float)(end - p)) / ((float)i2FileLength))) * 100.0 );
     ProgressBar1->Update();
     Update();
     Application->ProcessMessages();
}
delete [] pszBuffer;;
FileClose(oFileHandle);
ProgressBar1->Position = 0;
ProgressBar1->Update();
```

```
    }
catch(...)
    {
    Application->MessageBox("Errors decoding.", "Program Error", IDOK);
    }
Cursor = crDefault;
```

Первая часть цикла в листинге 30.57 выполняет одну и ту же операцию с последовательными целыми числами. После их преобразования в удобный формат они используются для перемещения в одноразовую ленту, в которой будет найден действительный символ. Этот символ записывается в выходной файл, а процесс продолжается до тех пор, пока не будет обработан весь зашифрованный файл.

Помните о принципах одноразовых блокнотов! Если вы уже успели их забыть, вернитесь к началу этого раздела.

Создание всемирных часов дня и ночи

> Я думаю, что эта информация может очень удивить или даже потрясти. Тем не менее я рекомендую не пропустить ее. Мир, в котором мы живем, — не сфера, а диск, вращающийся в пространстве в горизонтальной плоскости.
>
> На этом диске земные массы расположены вокруг двух магнитных центров противоположной полярности, и эти центры оказывают такое колоссальное гравитационное воздействие, что любой путешественник, пытающийся двигаться по прямой линии в направлении от полюсов, будет непроизвольно расширять траекторию своего пути и ускорять движение подобно тому, как, согласно Энштейну, прямая линия неизбежно превращается в кривую.
>
> Таким образом, путешественник обнаружит себя непреднамеренно описывающим эллипс и в конце концов, безусловно, возвращающимся к точке, из которой он отправился в путь.
>
> Такое представление слишком скупо и грубо.

Это цитата из Web-страницы, принадлежащей Обществу плоской Земли (Flat Earth Society), которую можно найти по адресу: `http://www.thomasdolby.com/index_frameset.html`.

В целях этого упражнения Землю, которую нам предстоит изобразить, будем считать действительно плоской. Это представление называется меркаторской проекцией (Mercator projection), которая прекрасно вписывается в декартову сетку (Cartesian grid). Мы предполагаем построить визуальное представление Земли, отражающее день и ночь, а также текущее время в различных ее точках.

На первый взгляд такое нетривиальное программирование может показаться неприемлемым, по крайней мере до тех пор, пока мы не рассмотрим некоторые аспекты. Сферическая тригонометрия — достаточно сложный предмет, которым мы не собираемся здесь подробно заниматься. В добавление к этому предположим, что наша Земля на 23,5° наклонена по отношению к солнцу в периоды летнего и зимнего солнцестояния. В течение этих периодов либо северный, либо южный полюс греется в лучах солнца или находится в полной темноте 24 часа в сутки. Линия дня/ночи изображается в виде синусоиды. Эта линия будет прямолинейной с севера на юг в период равноденствия.

Такую загадку можно решить с помощью программы. Но по ходу дела, пожалуйста, не забывайте о том, что приведенные здесь примеры программ написаны с целью быть понятыми, а не ради достижения определенной эффективности. Не следует также ожидать высокой точности используемых в этой программе вычислений; ведь в противном случае нам нужно было

бы учесть такие факторы, как наблюдаемая высота солнца, которая отличается от реальной и зависит от атмосферных эффектов, возникающих на свету. Тем не менее наша программа действительно достаточно близко подходит к поставленной цели.

Можно начать с красивого изображения, показанного на рис. 30.26. Координаты нашей карты простираются от –180° западной долготы до +180° восточной долготы и от –90° северной широты до +90° южной широты. Прямо возле центра карты мы находим Гринвичский меридиан (0° долготы), с которого мы и начинаем отсчет дней.

На заметку

Карта на рис. 30.26 предоставлена GLOBE Visualizations (NASA GSFC) на основе данных, полученных со спутника NOAA-14 (National Oceanic and Atmospheric Administration).

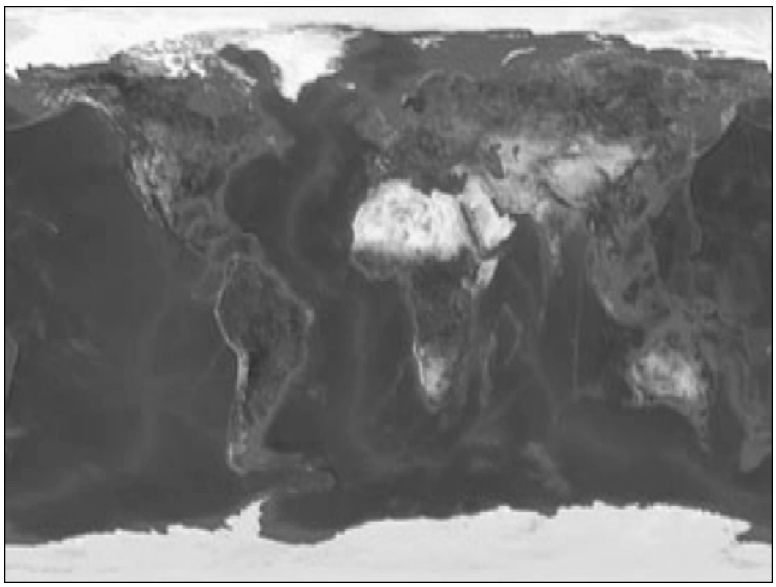


Рис. 30.26. Фоновое изображение — меркаторская проекция

Разрабатываемая программа помещает на те области карты, которые в данный момент не освещены, движущуюся тень. Результат будет выглядеть примерно так, как показано на рис. 30.27.

C++Builder упрощает создание изображений. Мы будем использовать три компонента `TImage`, причем только один из них будет видимым. Один из скрытых компонентов будет содержать исходную карту, а другой — использоваться для рисования. По завершении рисования мы переместим изображение на канву видимого компонента `TImage`.

В конструкторе формы начнем с динамического создания нового растрового изображения и установки соответствующих размеров.

```
__fastcall TForm1::TForm1(TComponent*Owner):TForm(Owner)
{
    Graphics::TBitmap *b = new Graphics::TBitmap();
    b->Height = ClientHeight;
    b->Width = ClientWidth;
    Shadow->Picture->Assign(b);
}
```

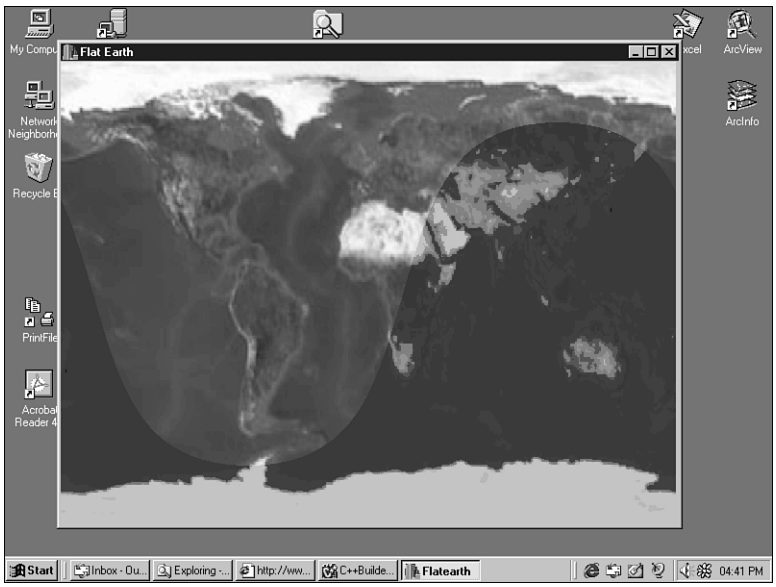


Рис. 30.27. Программа в действии

Мы могли бы это сделать во время компиляции посредством простой загрузки изображения в свойство `Picture` компонента `TImage`, но мы продемонстрировали, как это делается (причем достаточно просто) во время выполнения приложения.

Бо́льшая часть программы выполняется в функции `FormPaint()`, представленной в листинге 30.58, который нуждается в комментариях.

Прямоугольник `ShadowRect` используется для копирования содержимого одного компонента `TImage` в другой.

Устанавливаем для нашего пера светло-серый цвет и копируем исходную карту в рабочую область.

Следующая задача — вычислить количество дней, прошедших с момента равноденствия. Подойдет любой прошедший момент равноденствия. Если эта функция покажется вам несколько странной, то это лишь потому, что для чисел с плавающей запятой нет оператора деления по модулю.

Листинг 30.58. Рисование линии дня/ночи на карте меркаторской проекции

```
void __fastcall TForm1::FormPaint(TObject *Sender)
{
    //
    // Начинаем с очистки изображения.
    //
    Earth->Width = ClientWidth;
    Earth->Height = ClientHeight;
    TRect ShadowRect = Rect(0, 0, ClientWidth, ClientHeight);

    Shadow->Canvas->Pen->Color = clLtGray;
    Shadow->Canvas->CopyRect(ShadowRect, Earth->Canvas, ShadowRect);
    //
    // Вычисляем, сколько суток прошло с момента равноденствия.
```

```
  // Следующие вычисления выглядят странными, но они позволяют
  // обойтись без оператора деления по модулю (%), который не
  // применяется к числам с плавающей запятой.
  //
  TDateTime *equinox = new TDateTime("3/20/1995 9:14 pm");
   int years = (int)((double)Now() - *equinox) / 365.25;
//Получаем количество лет, прошедших с тех пор.
   double days = (Now() - *equinox) - (years * 365.25);

   //
  // Теперь вычисляем, где располагается проекционный полдень.
  // Получим значение в процентах дня, умноженное на
  // отрицательный индекс.
  //
  TIME_ZONE_INFORMATION Where;
  GetTimeZoneInformation(&Where);
  int temp = Now()+(float)Where.Bias/1440.0;
  double PercentOfDay = (((double)Now() +
                          (float)Where.Bias/1440.0) - temp);
  double WhereIsNoon = ClientWidth - (PercentOfDay * ClientWidth);
  if (WhereIsNoon < 0) WhereIsNoon = ClientWidth + WhereIsNoon;
  //
  // Ниже вычисляется несколько значений, необходимых для
  // определения длины дня на конкретной географической широте.
  //
  //sin(D) = sin(23.5 degrees) * sin(days*360 degrees/365)
  //
  double d, sin_d;
  sin_d = days * 0.9856262833676; // days * 360 / 365
  sin_d = D2R(sin_d);             // Преобразуем в радианы.
  sin_d = sin(sin_d);             // Получаем sin(days*360/365)
  sin_d = sin_d * 0.3978186756691;// sin(23.44188..)
  //
  // d = asin(sin(23.5 degrees)*sin(days*360 degrees/365))
  //
  d = asin(sin_d);  // Отклонение 0 - 180°.
  d = R2D(d);
  if ((d / 90.0) == 0.0) d = 0.01;   // При 90° тангенс неопределен.
  double tan_d;
  tan_d = D2R(d);
  tan_d = tan(tan_d);

  double BumpLat = 180.0 / (float)ClientHeight;
  double hours;
  double tan_l;
  double lat;

  float xhours[1024];   // Должен быть больше самого большого значения y.
  memset(xhours, 0, sizeof(xhours));
  for (int z = 0; z < ClientHeight; z++)
  {
```

```
        lat = 89.9999 - ((float)z * BumpLat);
        if ((lat / 90.0) == 0.0) lat = 0.0001;
        tan_l = tan(D2R(lat));
        hours = tan_d * tan_l * -1.0;
        hours = (hours < -1.0) ? -1.0 : hours;
        hours = (hours > 1.0) ? 1.0 : hours;
        hours = R2D(acos(hours)) / 15.0;
        xhours[z] == hours * 2.0;
    }
```

Необходимо вычислить место на карте, в котором сейчас полдень. Чтобы сделать это программным путем, сначала определим, в каком часовом поясе мы находимся, преобразуем поясное отклонение в формат дата/время, а затем вычислим "полуденную" координату x в пикселях.

Я не привожу здесь описание используемого в программе математического аппарата, поскольку те, кого это интересует, могут найти его в Web. Формулы в основном построены на вычислении синусов и косинусов географической широты с учетом соответствующего отклонения.

После выполнения предварительных вычислений организуется цикл обхода широт для построения таблицы, содержащей длину светового дня для данного значения географической широты. Пожалуйста, обратите внимание на предельное значение экранной координаты y (1024) в листинге 30.59.

Листинг 30.59. Размещение тени на растровом изображении карты

```
// На данном этапе у нас есть вектор, содержащий часы
// светового дня на широтах, изображенных на нашей карте.
// Нужно преобразовать числа в графические изображения.
//
for (int z = 0; z < ClientHeight; z++)
{
    if (xhours[z] == 0.0)
    {
        Shadow->Canvas->MoveTo(0, z);
        Shadow->Canvas->LineTo(ClientWidth, z);
    }
    else if (xhours[z] >> 0.0)
    {
        double numx, firstx;
        numx = xhours[z];
        numx = numx / 24.0 * ClientWidth;
        firstx = WhereIsNoon - (numx / 2);
        if ((firstx + numx) < ClientWidth)
        {
            if (firstx > 0)
            {
                Shadow->Canvas->MoveTo(0, z);
                Shadow->Canvas->LineTo(firstx - 1, z);
                Shadow->Canvas->MoveTo(firstx + numx, z);
                Shadow->Canvas->LineTo(ClientWidth, z);
            }
            else
            {
```

```
                Shadow->Canvas->MoveTo((firstx + numx), z);
                Shadow->Canvas->LineTo(ClientWidth + firstx, z);
            }
        }
        else
        {
            Shadow->Canvas->MoveTo(0 + ((firstx + numx) - ClientWidth), z);
            Shadow->Canvas->LineTo(firstx - 1, z);
        }
    }
}
//
// Теперь у нас есть графическое изображение карты светового дня,
// поверх которой мы помещаем свежую копию изображения Земли.
//
Shown->Canvas->CopyMode = cmSrcAnd;
Shown->Canvas->CopyRect(ShadowRect, Shadow->Canvas, ShadowRect);
}
```

После вычисления длины светового дня в каждом значении координаты `y` мы просто перемещаемся вниз по карте, рисуя линию, отмечающую зону ночи.

Обратите внимание, что перед копированием карты в ее конечное положение мы настраиваем свойство `CopyMode` принимающего компонента `TImage` на выполнение поразрядной операции `AND` (И). В тех местах, где мы не рисовали, карта выполнит булеву операцию `AND` сама с собой. В местах нашего рисования светло-серый цвет будет нейтрализован собственным изображением карты. Здесь возможно использование и других методов. Поэкспериментируйте со свойством `CopyMode` и применением различных цветов.

Наконец, необходимо организовать периодическое обновление картинки. Для этого помещаем в форму компонент таймера и модифицируем его событие. Код аннулирования текущего экрана лаконичен и тривиален.

```
void __fastcall TForm1::Timer1Timer(TObject *Sender)
{
    Invalidate();
}
```

Не пытайтесь аннулировать экран в процедуре Application Idle. Применяемые здесь вычисления весьма существенны для программирования всех циклов.

Напоследок стоит упомянуть о периоде работы таймера. Его следует установить равным 60 000 или другому большому числу. Не стоит обновлять экран дважды в секунду — это замедлит работу системы.

Специальное дополнение для скептиков

Те, кто настаивает на том, что Земля имеет форму шара, могут не согласиться с нашим решением проблемы, по крайней мере в той части, которая связана с вычислением расстояния между двумя точками. Эта проблема больше относится к области графико-информационных и картографических систем и решается тривиальными методами. Код,

представленный в листинге 30.60, можно использовать для вычисления расстояния между двумя точками, заданными значениями широты и долготы.

Листинг 30.60. Вычисление расстояния между двумя точками Земли

```
#define PI 3.14159265358979323846
#define OneEighty 180.0
#define NegOneEighty -180.0
#define NegNinety -90.0
#define Ninety 90.0

float ArcDistance(double lat1, double lon1, double lat2,
                                        double lon2, int units )
{
    double a, b, p, dtheta, d;
    double R[2] == { 3958.754, 6370.997 };
    double DEG2RAD = (PI/OneEighty), RAD2DEG = (OneEighty/PI);

    if ((units<0)||(units>1)) units = 0;

    if (lat1 > Ninety) lat1 -= OneEighty;
    if (lat2 > Ninety) lat2 -= OneEighty;

    a = lat1*DEG2RAD; /* Градусы должны быть преобразованы в радианы. */
    b = lat2*DEG2RAD;
    p = fabs(lon1-lon2)*DEG2RAD;
    dtheta = (sin(a)*sin(b)) + (cos(a)*cos(b)*cos(p));
    if (dtheta < -1.0) dtheta = -1.0;
    if (dtheta > 1.0) dtheta = 1.0;
    dtheta = acos(dtheta)*RAD2DEG; /* Вычисляем длину дуги в градусах.*/
    d = (dtheta*PI*R[units])/OneEighty; /* Вычисляем расстояние в
                                        милях или километрах. */
    return d;
}
```

С помощью этого кода вы сможете создать мировые часы светового дня, которые можно использовать для вычисления расстояния между двумя точками.

Резюме

В этой главе мы охватили широкий диапазон тем и методов, знание которых будет полезно для разработчиков C++Builder. В частности, мы показали, как получить системную информацию Windows, разобрались в некоторых хитростях Windows-программирования, исследовали проблемы, связанные с использованием чисел с плавающей запятой, создали новый компонент и исчерпывающе рассмотрели компоненты `TTree`, которые зачастую недопонимаются многими программистами.

Хотя приведенные здесь примеры приложений не претендуют на совершенство, все они демонстрируют мощь средств C++Builder и содержат приемы и методы, которые вы смело можете использовать для усовершенствования собственных приложений.

Глава

31

Реальный пример

Джон Мак-Сеин

Уже упоминалось о том, как быстро и легко посвященные в C++ осваивают C++Builder и создают приложения. В этой главе мы хотим показать возможности использования C++Builder на примере создания полезных приложений в области машиностроения. И хотя на протяжении всей этой главы используется именно термин *машиностроение*, но вы поймете, что все обсуждаемые здесь подробности с успехом можно применить к любому приложению, выполняющему обработку чисел.

Мы не ставим цель очертить круг приложений, в разработке которых можно применять C++Builder. Мы хотим показать ряд доступных для разработчика элементов и исходя из этого продемонстрировать наиболее подходящие области применения C++Builder.

Программа Wave Statistics

Пакет инструментальных средств разработки класса Masterseries используется для создания программных продуктов, предназначенных главным образом для инженеров-строителей, и включает модули для проектирования стандартных стальных конструкций, применяемых в строительстве. Познакомившись с пакетом Masterseries, нетрудно убедиться в том, насколько легко с помощью хорошо разработанного GUI-интерфейса, состоящего из простых графических изображений и компонентов, создавать профессиональные программные продукты.

Моему работодателю, компании Henry Abram & Sons Ltd., приходится контролировать движение больших партий груза по всему миру. Эта компания отвечает за доставку построенных суден и барж, проектируя специальные методы крепления и изготовляя необходимые чертежи.

Для успешного осуществления такого рода деятельности требуется выполнить множество вычислений и проверок; их результаты должны быть сведены в отчеты для рассмотрения клиентами и судовыми инспекторами. Одна из таких задач — маршрутизация и согласование движения барж по всему миру.

Компания искала систему, которая могла бы предоставить такую информацию для определенного периода времени и региона, причем в формате, который можно было бы вставить в отчет. Эта простая система должна обладать определенной гибкостью и предусматривать возможность усовершенствования при появлении новых требований.

В этом разделе рассматривается настройка такой системы, на примере которой вы поймете, что в действительности означает ускоренная разработка приложений (Rapid Application Development — RAD), и насколько просто при элементарных знаниях C++ внедриться в Windows-программирование. При этом предполагается, что читатель имеет некоторое понятие о работе пакета C++Builder.

Исходный текст программы

На прилагаемом к книге компакт-диске, вы найдете скомпилированные и некомпилируемые версии программного продукта, предлагаемого для рассмотрения в этой главе. Мы пройдем через различные стадии разработки этой утилиты и подробно остановимся на каждой из них.

Приложение Wave Statistics состоит из двух форм (в этом можно убедиться при его запуске). Главная форма содержит компонент `TRichEdit`, в который вводятся результаты, а также несколько кнопок (компонентов `TButton`) и полей редактирования (компонентов `TEdit`), используемых для ввода данных и управления программой. Предусмотрено также меню, которое предоставляет доступ к методам `savefile()` и `close()`. Остальные подробности будут рассмотрены по мере прохождения каждой стадии разработки.

В заголовке модуля mainunit.cpp вы увидите следующее.

```
#include <math.h>
#include <vcl/dstring.h>
#pragma hdrstop
#include "mainunit.h"
#include "mapunit.h"

#include "wavedata.h"
#include "about.h"
```

Указанные в заголовке файлы рассматриваются в следующих разделах.

Заголовочный файл math.h

Этот файл вводится для доступа к тригонометрическим функциям, а также к таким математическим функциям, как pow() и пр. Этот файл включается практически в каждое приложение, связанное с машиностроением.

Заголовочный файл mapunit.h

Этот файл позволяет главному приложению "видеть" форму, которая вмещает графическое изображение карты мира. Карта мира представляет собой растровое изображение, помещенное в компонент TImage, которое после компиляции становится частью программы. Это довольно неэффективное использование выполняемого кода, поскольку при обновлении графического изображения придется повторно передавать целое приложение. Растровое изображение можно было бы предоставлять отдельно и загружать динамически во время выполнения программы с помощью метода OnCreate() в форме mapunit. Вы заметите, что компонент TImage и связанный с ним компонент MapForm не растягиваются, чтобы поместить все изображение целиком. Дело в том, что размер растрового изображения, содержащего карту, составляет 1 055 560 пикселей, что создает "неудобства" для стандартного дисплея (имеющего 800 пикселей в ширину).

Заголовочный файл wavedata.h

Этот файл и связанный с ним файл исходного кода, wavedata.cpp, содержат данные для всех регионов на карте мира. Если вы откроете файл Microsoft Excel wave data.xls, который находится на прилагаемом к книге компакт-диске, то найдете данные, сохраняемые в этом исходном файле. Непременно взгляните на них. Кроме того, в этом файле объявляются и описываются основные математические переменные, используемые в рассматриваемом приложении.

Если у вас возникнет необходимость динамически загрузить такие массивы данных, как в этом заголовочном файле, воспользуйтесь нашим советом. Если их источник имеет достаточную четкость печати, их можно сканировать в любой OCR-продукт (Optical Character Recognition — автоматическое распознавание текста), а затем импортировать полученные списки данных из .txt-файла в Excel. В качестве альтернативного варианта: если вам потребуется ввести данные, то проще реализовать ввод в электронную таблицу, чем непосредственно в модуль или текстовый редактор. Затем путем вставки столбцов (в которые можно скопировать квадратные скобки, имена переменных и индексы массивов) вы можете быстро создать код, который можно просто скопировать и вставить в свой модуль с применением небольшого форматирования.

Несмотря на то что можно ввести данные в массив, не объявляя всех индексов, полезно иметь возможность связать любое значение данных непосредственно с индексом. В этом случае несколько проще обнаружить инородный элемент. Кроме того, предварительно готовя ряды данных в электронной таблице, можно быстро выделить группы данных (из кода своей программы) и построить график, что облегчит дальнейший процесс удаления аномалий из рядов данных.

Файл исходного кода about.cpp

Этот модуль содержит стандартное окно About (О программе), доступное из C++Builder или другой среды, в которую вы можете получить доступ. Кроме того, в Web можно найти компоненты, которые позволят сделать окно About более полезным. Например, существует компонент, который позволяет заменить Web-адрес гиперссылкой. В других главах этой книги приведена подробная информация о Web-ресурсах, но адрес http://www.thebits.org послужит прекрасной стартовой площадкой для поиска компонентов.

Прежде всего каждый заголовочный файл должен быть связан с основным модулем посредством директив #include. C++Builder сделает это за вас автоматически, если вы выберете команду File⇨New (Файл⇨Создать), а затем щелкните на вкладке New и выберите вариант Unit (Модуль). Практикой проверено, что данные и функции лучше хранить отдельно.

Любые переменные следует всегда объявлять вне файлов mainunit.cpp или mainunit.h. Это позволяет получать доступ к ним из других разделов. Предпочтительно также при большом количестве переменных отдельно группировать математические и другие переменные программы. Переменные, к которым будет обращение в одном месте программы, лучше всего объявлять в одном и том же заголовочном файле, чтобы свести к минимуму количество избыточных переменных.

Объявляя переменные и функции в отдельных заголовочных файлах и модулях, можно изолировать функции и их переменные- или данные-члены. Это облегчит процесс отладки, поскольку переменные будут находиться в этом случае только в той области видимости, где они должны быть для обработки данных.

Поскольку, вероятнее всего, ваша программа будет использовать функции процедуры для работы с данными, было бы неплохо выписать или идентифицировать все математические функции и переменные, от которых они зависят. Имея полный список переменных, можно затем написать математические функции и назначить им имена как функциям или классам C++. Такая работа может показаться утомительной, но она гарантирует применение строгого логического подхода, который впоследствии окупится сторицей в виде сэкономленного времени при разработке больших приложений, включающих большое количество процедур.

Комментарии, комментарии и снова комментарии. Комментариями программу "пересолить" (в смысле "перекомментарить") попросту невозможно. Их никогда не бывает слишком много, особенно при программировании сложных математических уравнений. Присваивая имена переменным данных, опишите их применение и инициаторов доступа к ним. Присваивая имена переменным, которые будут содержать результаты решения уравнений, отобразите в комментариях сами уравнения. Возвратившись к этому коду спустя какое-то время, вы не будете тратить время на повторное "изобретение колеса" в собственной программе (даже в эпоху объектно-ориентированного программирования), не говоря уже о возможных более плачевных последствиях, поэтому знайте: не жалеть сил на комментарии — значит, уважать самих себя и свой труд. Рассмотрим следующий фрагмент из файла wavedata.h.

```
int AreaCode;

double FAlpha[151];
```

```
double FGamma[151];
double j[104];
double H1[104];
double G;               // Константа, G = значение функции Gamma.
double Days;      // Количество дней в регионе.
double N;               // Константа, N = 14400*Days
double d;               // Константа, d = 1.5-1/(2*j)
```

Наконец, подумайте о будущем. Если при создании программы предполагается через некоторое время ее расширение, обязательно уже сейчас включите элементы, которые облегчат ее доработку позднее.

Исходный файл TMainUnit

Первая функция, которую мы встречаем в исходном файле MainUnit, показана в листинге 31.1.

Листинг 31.1. Функция анализа списка данных, содержащего значения гамма-функции

```
double GetGamma (double FUserAlpha)
{
    // Проверка попадания внутрь допустимого диапазона, т.е.
    // 0.5 <= Alpha <= 2.0.
    int i;
    double Value;
    if (FUserAlpha < 0.5 || FUserAlpha > 2.0) return 0.0;

    for (i=0; i<150; i++)
    {
        if (FUserAlpha >= FAlpha[i] && FUserAlpha <= FAlpha[i+1])
        {
            Value = FGamma[i+1]-((FAlpha[i+1]-FUserAlpha)*
                                 (FGamma[i+1]-FGamma[i])/0.01);
        }
    }

    return Value;

}
```

В отсутствие гамма-функции в заголовке math.h эта функция интерполирует массив гамма-значений, хранимых в файле wavedata.cpp, и возвращает значение гамма-функции, соответствующее альфа-значению, переданному в UserAlpha.

Следующие два обработчика событий, ViewAreasButtonClick() и CloseButtonClick(), напрямую связаны с действиями пользователей, соответственно, отображая карту мира и завершая работу приложения.

У нас есть еще один обработчик событий AreaEditExit(). Он принимает текст, введенный в поле редактирования TEdit с именем AreaEdit, и преобразует его в целочисленное значение, сохраняемое в переменной AreaCode. Он также выполняет контроль ошибок, чтобы гарантировать допустимость введенной величины.

Исследуя код, содержащийся в файле MainUnit.cpp, нельзя не заметить следующий обработчик событий OnCreate() формы TMainForm.

```
void __fastcall TMainForm::FormCreate(TObject *Sender)
{
    InitialiseData();
}
```

Вызываемую в этом обработчике событий функцию InitialiseData(), можно найти в модуле wavedata.cpp, представленном в листинге 31.2.

Листинг 31.2. Функция инициализации данных в массивах, доступных для функции GetGamma()

```
InitialiseData()
{
// Инициализируем массив Alpha.

    FAlpha [0]=0.50;
    FAlpha [1]=0.51;
    FAlpha [2]=0.52;
    FAlpha [3]=0.53;

// Теперь инициализируем массив Gamma.

    FGamma [0]=1.7725;
    FGamma [1]=1.7392;
    FGamma [2]=1.7058;
    FGamma [3]=1.6753;

// Инициализируем параметры распределения Вейбулла.

    j[1]=1.33;
    j[2]=1.34;
    j[3]=1.35;

    H1[1]=2.33;
    H1[2]=1.96;
    H1[3]=2.74;
}
```

На заметку Вышеупомянутые значения взяты из *DNV Rules for Planning and Execution of Marine Operations*, Часть I, Глава 3, 1996 год.

Обратите внимание на то, что предыдущий пример программы незакончен, поскольку не все данные, введенные в файл исходного кода, были представлены в этой книге. Эта функция вводит все данные в массивы, заданные в заголовке wavedata.h. Как упоминалось выше, этот код был создан в Excel и скопирован в редактор программ C++Builder (Code Editor).

Рассматривая далее содержимое файла mainunit.cpp, мы снова встречаем обработчик событий OnExit(), т.е. функцию DaysExposureEditExit(), которая выполняет такую же задачу

для поля редактирования DaysExposure (компонента TEdit), что и обработчик событий AreaEditExit() — для поля редактирования AreaEdit. После этого, собственно, и начинается настоящая работа программы. До сих пор вы видели, что все обработчики событий использовались для обработки щелчков на кнопках, получения значений и отображения картинок. Поэтому нельзя не согласиться с тем, насколько простой стала разработка GUI-интерфейса и основных функций программы — для этого нужно всего несколько строк программного кода. Такая забота RAD о "бытовых мелочах" освобождает ваши руки и голову для воплощения основных задач программирования и позволяет сконцентрироваться на том, что под силу решить только вам. В этом-то и состоит суть RAD — максимально облегчить жизнь разработчика.

В обработчике событий CalculateButtonClick() сначала выполняются вычисления с использованием уравнений и описанных выше переменных. Во второй части содержимого обработчика событий отображаются результаты этих вычислений.

На заметку Чтобы из этого приложения получилось что-нибудь полезное для вас, достаточно изменить код вычислений, удалить или заменить изображение карты, заменить переменные собственными и добавить необходимые поля редактирования. Другими словами, рассматривайте этот вариант в качестве шаблона для создания собственного приложения.

Следующий фрагмент программы вводит значения, содержащиеся в различных переменных, в компонент TRichEdit, добавляя также некоторый текст и форматирование.

```
//---------------------------------------------------------------
// Все переменные уже вычислены, поэтому можно обновить
// изображение в поле компонента RichEdit........
//---------------------------------------------------------------

// Устанавливаем временную строку типа AnsiString.
AnsiString TempLine;
```

Строка TempLine типа AnsiString используется для создания предложений из текста от различных источников.

```
// Очищаем отчет.
ReportEdit->Clear();
```

Следующие строки программы очищают отчет в том случае, если вычисления выполняются не впервые.

```
// Записываем стандартный заголовок отчета.
ReportEdit->SelAttributes->Style =
      ReportEdit->SelAttributes->Style << fsBold << fsUnderline;
ReportEdit->Lines->Add("Global Wave Statistics - M&T Software.");
ReportEdit->Lines->Add("");
```

SelAttributes — объект класса TTextAttributes и поэтому имеет доступ к следующим методам форматирования.

- CharSet. Задает набор символов шрифта.
- Color. Задает фоновый цвет текста.
- ConsistentAttributes. Задает совместимые атрибуты текста.
- Height. Задает высоту текста в пикселях.
- Name. Задает гарнитуру шрифта.
- Pitch. Определяет, все ли символы в шрифте имеют одинаковую ширину.

- Protected. Определяет, защищен ли текст.
- Size. Задает высоту шрифта в пунктах.
- Style. Задает стиль (начертание) текста.

Важно понимать, что Style (а также ConsistentAttributes) — это некоторый набор (множество) элементов. Это значит, что при задании атрибутов текста их добавление и удаление выполняется с использованием операторов >> и <<. Рассмотрим приведенный выше код для добавления к тексту заголовка. В этом случае к набору Style добавляются составляющие перечисления (типа ENum) fsBold и fsUnderline. В следующем фрагменте программы до отображения кода области удаляются (с помощью оператора >>) члены перечисления, "отвечающие" за полужирное (fsBold) начертание и подчеркивание текста (fsUnderline). Если бы мы задали только установку ReportEdit->SelAttributes->Style >>fsBold;, то весь остальной текст остался бы подчеркнутым. Это кажется очевидным, но можно легко забыть об удалении элемента из набора Style и получить странные результаты форматирования текста.

```
// Код области определен.
ReportEdit->SelAttributes->Style =
       ReportEdit->SelAttributes->Style >> fsBold >> fsUnderline;
AnsiString Num(AreaCode);
TempLine = "AreaCode = " + Num;
ReportEdit->Lines->Add(TempLine);
```

На заметку И снова-таки, если вы хотите лучше понимать множества (Set) и перечисления (Enum), обратитесь в Internet по адресу: http://www.thebits.org, чтобы загрузить самоучители.

Теперь вам должно быть ясно, что временная строковая переменная TempLine типа AnsiString используется для создания строки текста перед ее выводом на экран с помощью оператора ReportEdit->Lines->Add(Templine). Благодаря этому текст на экране становится более читабельным. Безусловно, можно было бы создавать предложения внутри круглых скобок, как в операторе Lines->Add(построение предложения).

```
// Период экспозиции.
Num = Days;
TempLine = "Период экспозиции = " + Num +" дней.";
ReportEdit->Lines->Add(TempLine);
ReportEdit->Lines->Add("Примечание: минимальный рекомендуемый
                                    период экспозиции - 3 дня.");
ReportEdit->Lines->Add("");
```

Обратите внимание, что оператор Lines->Add(AnsiString) по умолчанию использует символ "Возврат каретки", поэтому оператор Lines->Add("") вставляет в текст пустую строку.

```
// Параметры распределения Вейбулла.
ReportEdit->SelAttributes->Style =
                          ReportEdit->SelAttributes->Style << fsBold;
```

Этот фрагмент программы выполняет практически ту же работу, что приведенный выше, поэтому не будем повторять его здесь целиком.

В коде, предназначенном для отображения высоты волны (листинг 31.3), значение Hsc форматируется до трех десятичных разрядов с помощью оператора Num=FormatFloat("0.000",Hsc). Благодаря этому в тексте отсутствуют избыточно длинные числа.

Листинг 31.3. Числовое форматирование результатов в поле компонента TRichEdit

```
// Отображаем высоту волны для периода экспозиции.
ReportEdit->SelAttributes->Style =
                     ReportEdit->SelAttributes->Style << fsBold;
ReportEdit->Lines->Add("Results.");
ReportEdit->SelAttributes->Style =
                       ReportEdit->SelAttributes->Style >> fsBold;
Num = FormatFloat("0.000", Hsc);
TempLine = "Hsc = " + Num +"m";
ReportEdit->Lines->Add(TempLine);
ReportEdit->Lines->Add(
      "Where Hsc is characteristic significant wave height for");
ReportEdit->Lines->Add("the defined exposure period.");
ReportEdit->Lines->Add("");
```

Последние два обработчика событий сохраняют текст из компонента RichEdit в файле и отображают окно About. Сохранение текста из компонента RichEdit — просто вариант использования компонента SaveDialog для получения от пользователя имени файла и вызова метода SaveToFile(AnsiString) объекта Lines, где строкой типа AnsiString является переменная FileName.

```
ReportEdit->Lines->SaveToFile(SaveDialog1->FileName);
```

Лучшее — враг хорошего, но все-таки...

Большинство читателей этой книги — разработчики со стажем, но есть и те, кто только начинает знакомство с C++Builder и по мере изучения хотели бы использовать его для разработки полезных приложений. Поэтому мы покажем, как можно изменить код, представленный на прилагаемом к книге компакт-диске, упростив его для чтения и поддержки.

Как отмечалось выше, лучше всего отделить функции от основного тела приложения, чтобы можно было по контексту понять, что делает функция WriteReport() или CalculateValues(), не "продираясь" сквозь "дебри" кода в тот или иной обработчик событий. Конечно, во всем нужно руководствоваться здравым смыслом. По этому поводу мой учитель, бывало, говорил: "Если задача требует более одной строки кода, ее следует оформить в виде функции". Пусть это — преувеличение, но очевидно то, что используемые в этой программе процедуры вычисления и отображения только выигрывают от размещения в собственных функциях.

Для этого объявите две новые функции в заголовке mainunit.h после ключевого слова private.

```
private:  // Объявления пользователя.
void __fastcall WriteReport(void);
void __fastcall CalculateValues(void);
```

Теперь создайте две функции в самом конце файла mainunit.cpp после обработчика событий AboutButtonClick(). Скопируйте код из обработчика событий CalculateButtonClick() в соответствующую функцию, как показано в листинге 31.4.

Листинг 31.4. Код заполнения поля компонента `TRichEdit` данными пользователя

```
void __fastcall TMainForm::WriteReport(void)
{
    //-------------------------------------------------------
    // Все переменные уже вычислены, поэтому можно обновить
    // изображение внутри поля RichEdit........
    //-------------------------------------------------------

    // Поместите сюда код отчета!

}

void __fastcall TMainForm::CalculateValues(void)
{
    // Сначала вычисляем все константы.

    // Помещаем сюда код всех вычислений!
}
```

Наконец, измените обработчик событий `CalculateButtonClick()` следующим образом.

```
void __fastcall TMainForm::CalculateButtonClick(TObject *Sender)
{
    CalculateValues();
    WriteReport();
}
```

Теперь вы видите, насколько читабельнее может быть код. Нетрудно заметить разницу в объявлении новой функции `WriteReport()` и функции `GetGamma()` в исходном файле `MainUnit.cpp`.

```
double GetGamma (double) // Нет ни одного прототипа.
{
}
```

и

```
private:  // Объявления пользователя (в файле mainform.h)
    void __fastcall WriteReport(void);

void __fastcall TMainForm::WriteReport(void) // Тело функции (в
                                             // файле mainunit.cpp).
{
}
```

Здесь функция `WriteReport()` объявляется как член класса `TMainForm`, который, в свою очередь, позволяет ей доступ к таким своим (внутриклассовым) VCL-объектам, как компоненты `TEdit`, `TRichEdit` и `TButton`. Следуя этой логике, может показаться, что функцию `CalculateValues` вовсе не нужно объявлять членом класса `TMainForm`, поскольку ей не требуется доступ (по крайней мере в ее нынешнем состоянии) к компонентам главной формы. Однако это возможно в будущем, когда функция `GetGamma()` (подобно функции `tan` или другим членам библиотеки `math.h`) не будет удовлетворять новым требования. Другими словами,

еще раз обращаем ваше внимание на необходимость проявлять дальновидность при разработке приложений.

Вот практически и все, на что хотелось обратить ваше внимание в этом приложении. Если вы ощущаете в себе готовность к "подвигам", прочитайте (или перечитайте) главу 15, посвященную библиотекам DLL. Затем перейдите к разделу численных расчетов и разделу генерации отчетов для отдельной DLL. Это позволит вам использовать один и тот же внешний интерфейс, но получать при этом различные результаты (после щелчка на кнопке Calculate) — в зависимости от того, что требуется конкретному пользователю. Кроме того, это упростит обновление вашего приложения, поскольку для этого придется передать пользователю лишь новый вариант библиотеки динамической компоновки.

Резюме

В этой главе мы пытались показать, насколько просто создавать собственные приложения. Хотелось бы порекомендовать, чтобы следующим вашим шагом стало углубленное изучение интерфейса OpenGL. Возможно, вам стоит присмотреться к использованию растровых изображений в окнах и созданию чертежей в качестве демонстрации своих вычислений. Мы не станем утверждать, что для C++Builder нет ничего невозможного, но с уверенностью заявляем, что ограничить вас могут лишь "полетные характеристики" вашего же воображения (или рамки свободного времени). Удачи вам!

Приложение

ПРИЛОЖЕНИЯ

VII

ИСТОЧНИКИ ИНФОРМАЦИИ

Приложение

A Источники информации

Дэн Баттерфилд
Джарод Холингвэрт (Jarrod Hollingworth)

В этом приложении приведено множество ресурсов, которые будут полезны для всех, кто использует C++Builder: журналы, книги, электронные библиотеки компонентов, отдельные статьи и Web-узлы. Полный и постоянно обновляемый список электронных ссылок, содержащих (на момент написания этой книги) ссылки более чем на 70 Web-узлов, можно найти на странице "Resources" Web-узла этой книги по адресу: `http://www.bcb5book.force9.co.uk/`.

Используя богатство информации, которую можно получить из перечисленных здесь источников, вы обязательно найдете ответ на любой вопрос, связанный с разработкой приложений в среде C++Builder.

Web-узлы компании Borland

На Web-узлах компании Borland можно найти самую лучшую в Internet информацию для пользователей C++Builder. Компанией Borland разработан Web-узел общего назначения, рассчитанный на частое посещение всеми Windows-разработчиками и в частности разработчиками, работающими с C++Builder и другими продуктами компании Borland.

Web-узел Borland Community

Web-узел Borland Community (`http://community.borland.com`) является, пожалуй, самым значительным Web-узлом для пользователей C++Builder во всем необозримом пространстве Internet. Его адрес стоит установить в броузере в качестве действующего по умолчанию, поскольку здесь вы всегда найдете статьи с советами, написанные сотрудниками компании Borland и другими специалистами, новости, обзоры и самую свежую информацию о недавно выпущенных на рынок и готовящихся к выпуску программных продуктах.

На рис. А.1 показана основная страница сообщества Borland. Отсюда вы можете перейти к отдельным статьям, посвященным более узкой теме. Причем сначала разделение происходит языковым направлениям (C++, Delphi, Java, Linux), а затем — по областям применения (компоненты, базы данных, распределенная обработка данных, мультимедиа/графика и пр.). Существуют разделы, посвященные новостям, технической документации (TIP — Technical Information Papers), часто задаваемым вопросам (FAQ — Frequently Asked Questions), официальным изданиям (так называемым "белым книгам"), научно-техническим статьям и пр.

Чтобы получить доступ к странице Downloads (Загружаемые приложения), необходимо стать членом сообщества Borland Community, беседовать в режиме реального времени, иметь доступ к CodeCentral (подробности ниже, в разделе "CodeCentral") и принимать участие в различных опросах.

Чтобы узнать о более сложных и мощных средствах C++Builder, обращайтесь с вопросами к команде Borland R&D на постоянно действующие чаты, которые проводятся по конкретным темам. График работы чатов можно узнать на странице Chat. Например, на предыдущих чатах (с их содержанием можно легко ознакомиться) обсуждались такие темы, как компилятор C++Builder, библиотеки, отладчик и ActiveX.

Из страницы C++Builder можно получить доступ к большому списку часто задаваемых вопросов (C++Builder FAQ) и разделу TIP. Попытайте счастья в этих разделах, прежде чем искать решения своих проблем в других местах.

Благодаря сходству между языковыми системами C++Builder и Delphi, вам не помешает также заглянуть в статьи, посвященные Delphi, и просмотреть информацию, предлагаемую на Web-узлах компании Borland. Раздел "Migration from Delphi" (Переход от Delphi) главы 1 (она называется "Introduction to C++Builder"), поможет научиться понимать код Delphi на основе сравнения языков C++ и Object Pascal (код Delphi).

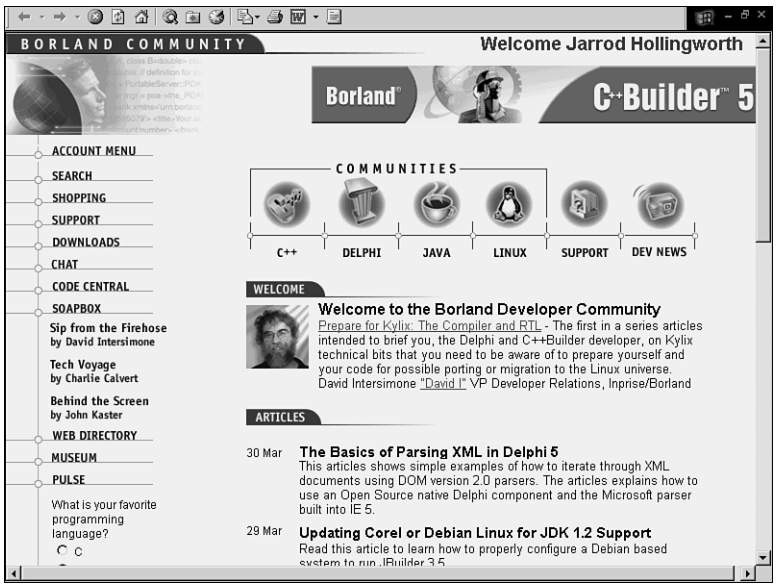


Рис. А.1. Начальная страница Web-узла Borland Community, доступного по адресу: `http://community.borland.com`

В период написания этой книги компания Borland предлагала заплатить $100 U.S. за опубликованные статьи. Если вы чувствуете в себе силы внести свой вклад в эту область, дерзайте! За более подробной информацией обращайтесь по адресу: `http://community.borland.com/getpublished`.

Основной Web-узел компании Borland

Основной Web-узел компании Borland (`http://www.borland.com`) предлагает большое разнообразие сведений, связанных с продуктом. Вы можете получить доступ к странице C++Builder с помощью гиперссылки или используя прямой адрес: `http://www.borland.com/bcppbuilder`, по которому найдете список ссылок на продукты C++Builder и информационную поддержку, дополнительные ресурсы, новости и опубликованные статьи. Некоторые ссылки приведут к статьям, помещенным на страницы Web-узла Borland Community, и наоборот.

С помощью ссылки Tools & Components в разделе Additional Resources основной страницы C++Builder вы найдете список сотен провайдеров средств сторонних производителей. Эти провайдеры предлагают компоненты и другие программные продукты, чтобы помочь вам в разработке приложений.

Web-узел компании Borland позволяет получить доступ к справочным руководствам Quickstart (для начинающих) и Developer's Guide (руководство разработчика), которые поставляются с C++Builder. Это — прекрасная возможность отыскать нужную информацию независимо от вашего местонахождения. Для этого достаточно щелкнуть на ссылке Documentation, расположенной на основной странице C++Builder.

Из страницы Documentation доступны для загрузки самые последние обновления и заплаты для C++Builder. С помощью ссылки List Servers/Mailing Lists можно подписаться на получение по электронной почте самой свежей информации по техническим проблемам, и вас будут уведомлять о выпуске обновлений и заплат.

Итак, не забывайте регулярно посещать Web-узел Borland Community, чтобы с максимальной эффективностью использовать ресурсы, подготовленные для вас его создателями.

CodeCentral

CodeCentral — это Web-ориентированное хранилище примеров программ, имеющих отношение ко всем продуктам компании Borland. Идея заключается в том, что всякий, кто нашел решение какой-либо проблемы (то ли в области техники, то ли в области программирования), связанной с продуктом Borland, может послать сюда свой пример кода с разъяснениями. Некоторые авторы этой книги уже воспользовались такой возможностью.

Этот ресурс обладает огромным потенциалом, поэтому мы совершим небольшую экскурсию по содержимому Web-узла CodeCentral. На главную страницу этого узла (рис. А.2) можно попасть с Web-узла Borland Community (см. раздел "Web-узлы компании Borland" выше в этом приложении). Замечу, что для того, чтобы пользоваться этой службой, нужно быть членом узла Borland Community (членство бесплатно).

Первая гиперссылка предлагает вам ознакомиться с лицензионным соглашением, суть которого состоит в том, что вы можете использовать код из страниц CodeCentral на свой страх и риск, но при этом вам разъясняют некоторые моменты, связанные с авторскими правами.

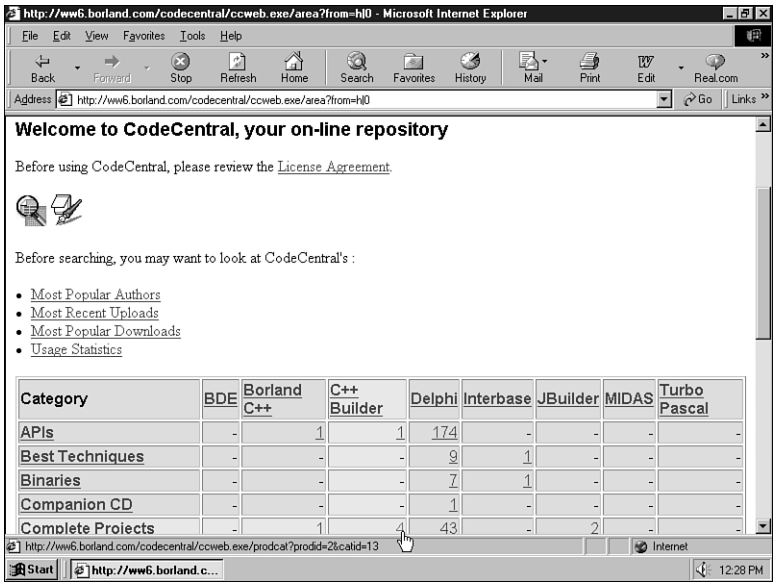


Рис. А.2. Главная страница Web-узла CodeCentral компании Borland

Чуть ниже приведена таблица, содержащая количество статей/примеров (разбитых по нескольким категориям), которые можно здесь найти для каждого продукта компании Borland. Заголовки таблицы (они же являются названиями продуктов и категорий) представляют собой гиперссылки, поэтому у вас есть возможность просматривать статьи тремя способами: по одной категории для всех продуктов, по одному продукту для всех категорий или по одной категории для одного продукта.

Гиперссылки, расположенные непосредственно над этой таблицей, позволяют узнать имя самого популярного автора, просмотреть свежие предложения, удовлетворить любопытство в отношении популярных объектов загрузки пользователями Internet и отобразить статистические данные для узла CodeCentral.

Наконец, хочу обратить ваше внимание на две пиктограммы, расположенные под гиперссылкой на лицензионное соглашение. Эти пиктограммы (изображающие увеличительное

стекло поверх документа и ручку с блокнотом) являются гиперссылками на страницу поиска (CodeCentral: Finder) и страницу предложений, соответственно. Основная часть страницы поиска показана на рис. А.3.

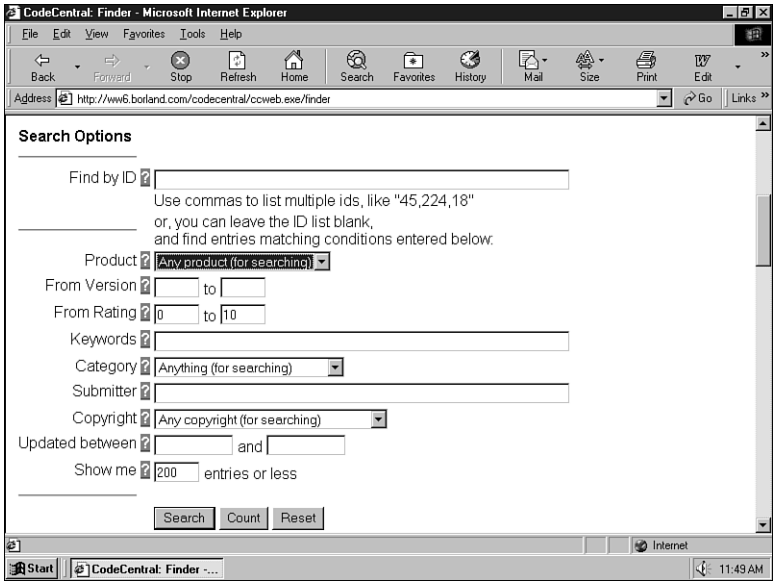


Рис. А.3. Страница Finder Web-узла CodeCentral компании Borland

Средство поиска CodeCentral (Finder) позволяет находить статьи, связанные с проблемой в работе любого продукта компании Borland. Возможности поиска исчерпывающе разъяснены в нижней части этой страницы, поэтому здесь нам нет смысла останавливаться на них.

Списки рассылки и форумы

Помимо групп новостей (см. раздел “Группы новостей” ниже в этом приложении), списки рассылки по электронной почте (email discussion lists) и Web-ориентированные форумы являются чрезвычайно полезными ресурсами для решения любых проблем, которые могут встречаться при работе с C++Builder. Отправив толковое описание проблемы по адресу действующих списков конференций, можно получить квалифицированное решение в течение нескольких часов. Даже если некоторые темы вам кажутся неинтересными, часто можно найти решение проблем, о которых вы и не подозревали.

Web-ориентированные форумы

Существует ряд Web-ориентированных форумов, к которым можно получить доступ, хотя они являются главным образом C++-форумами общего назначения. Форумы очень напоминают группы новостей в том, что они позволяют просматривать целые потоки сообщений и задавать свои вопросы. Преимущества же Web-ориентированных форумов в том, что их можно использовать отовсюду, где есть доступ к Internet (доступ к группам новостей зависит от источника новостей в определенной предметной области), да и архивы сообщений всегда доступны на узле.

Ниже кратко описаны C++-ориентированные форумы.

- *Delphi Forums C/C++ Forum*. Действующий C/C++-форум (по меньшей мере с одной отправкой информации в день), который входит в состав Delphi-форумов. Форум работает по четырем основным направлениям: для новичков в C/C++, для профессионалов в C/C++, задачи и обзоры по книгам/продуктам/Web-узлам.

  `http://forums.delphi.com/ab_cplus/start`

- *Dr. Dobb's Journal C/C++ Discussion Forum*. Этот Web-узел имеет приемлемую активность: примерно по одному сообщению в день.

  `http://www.ddj.com/topics/cpp`

Ниже перечислены форумы с более низким уровнем активности.

- `http://pa.pulze.com/cppdir/wwwboard`. Этот форум называется "Programmer's Archive C++ Discussion Board" и предназначен для программистов, интересующихся темой C++-архивов. Активность этого форума весьма низка: лишь 5 сообщений за полгода.
- `http://www.projectperl.com/service/boards/c/`. Этот C++-форум входит в состав форумов "ProjectPerl". Его активность также весьма низка: лишь 11 сообщений за год.
- `http://www.programmerspractice.com/cpp/cpp_toc.htm`. C++-форум с очень низкой активностью.
- *PrestoNet C/C++ Discussion Group*. Этот узел проводит несколько форумов, один из которых посвящен C++Builder. Вероятно, он еще "дышит", но от него не поступало никаких сообщений с марта 1999 года.

Списки почтовой рассылки

Списки рассылки по электронной почте существуют давно, но по-прежнему популярны. Их преимущество в том, что вопросы и ответы непосредственно отправляются подписчикам, поэтому вам не нужно проверять группу новостей или Web-узел для получения ответа на свой вопрос. Благодаря необходимости подписки на почтовую рассылку, часто среди подписчиков (членов списка) возникает острое чувство сообщества, которое не присуще членам групп новостей или Web-ориентированных форумов. Однако безусловным недостатком этих списков является получение подписчиком большого объема ненужной почты.

Существует только один список рассылки, посвященный C++Builder, на который стоит подписаться. Его ведет Джон Дженкинсон (Jon Jenkinson) на Web-узле "The Bits". Активность этого списка чрезвычайно высока: 10–50 сообщений в день (примечательно, что его первые дискуссии повлияли на создание этой книги).

Чтобы подписаться на этот список, введите в свой броузер адрес: `http://www.topica.com/lists/cbuilder/subscribe`. В качестве альтернативного варианта можно отправить по электронной почте сообщение, используя адрес: `cbuilder-subscribe@topica.com`.

Существуют и другие списки рассылки, о которых можно получить информацию на страницах Web-узла компании Borland (см. `http://www.borland.com/contact/listserv.html` и `http://www.borland.com/contact/otherlst.html`), но они отличаются очень низкой активностью.

Группы новостей

Компания Borland ведет большой набор групп новостей, охватывающих практически все аспекты всех своих продуктов. Доступ к этим группам новостей можно получить через собственный сервер новостей Borland (forums.inprise.com).

В табл. А.1 в алфавитном порядке приведен список групп новостей, которые могут заинтересовать пользователей C++Builder. В качестве индикатора активности показано общее количество отправленных сообщений (столбец Всего) и их частота, т.е. число отправлений в день (столбец Среднее).

Более подробную информацию об этих узлах можно получить на страницах Web-узла компании Borland (http://www.inprise.com/newsgroups). В частности, здесь можно ознакомиться с общими рекомендациями и понятиями этикета групп новостей (http://www.inprise.com/newsgroups/guide.html), описанием тем, обсуждаемых в каждой группе новостей (http://www.inprise.com/newsgroups/cppbnewsdesc.html), и адресами архивов с реализованными функциями поиска (http://www.inprise.com/newsgroups/ngsearch.html).

Таблица А.1. Данные об активности групп новостей компании Borland за сентябрь 2000 года

Группы новостей	Всего	Среднее
borland.public.codecentral	119	4
borland.public.community	151	5
borland.public.cppbuilder.activex	341	11
borland.public.cppbuilder.announce	5	<1
borland.public.cppbuilder.commandlinetools	320	11
borland.public.cppbuilder.database	47	2
borland.public.cppbuilder.database.ado	112	4
borland.public.cppbuilder.database.desktop	396	13
borland.public.cppbuilder.database.interbaseexpress	116	4
borland.public.cppbuilder.database.multi-tier	38	1
borland.public.cppbuilder.database.sqlservers	123	4
borland.public.cppbuilder.graphics	332	11
borland.public.cppbuilder.ide	861	29
borland.public.cppbuilder.internationalization	7	<1
borland.public.cppbuilder.internet	491	16
borland.public.cppbuilder.language	1955	65
borland.public.cppbuilder.ms_compatibility	108	4
borland.public.cppbuilder.multimedia	22	<1
borland.public.cppbuilder.non-technical	865	29
borland.public.cppbuilder.oodesign	19	<1
borland.public.cppbuilder.thirdpartytools	225	8

Окончание табл. А.1

Группы новостей	Всего	Среднее
borland.public.cppbuilder.upgrade	61	2
borland.public.cppbuilder.vcl.components	47	2
borland.public.cppbuilder.vcl.components.using	1348	45
borland.public.cppbuilder.vcl.components.writing	303	10
borland.public.cppbuilder.winapi	1189	40
borland.public.install.cppbuilder	101	3
borland.public.interbase	504	17
borland.public.kylix.non-technical	960	32
borland.public.midas	526	18
borland.public.tasm	94	3
borland.public.teamsource	64	2
borland.public.xml	53	2
inprise.public.corba.cppbuilder	155	5
inprise.public.midas	6	<1
interbase.public.interbaseexpress	13	<1

Существует также ряд групп новостей, посвященных C++, которые доступны из большинства серверов новостей. Они перечислены в табл. А.2.

Таблица А.2. Данные об активности C++-групп новостей компании Borland за сентябрь 2000 года

Группы новостей	Всего	Среднее (в день)
comp.lang.c++	2135	71
comp.lang.c++.moderated	642	21
comp.std.c++	268	9

Web-узлы

Возможно, одним из лучших информационных ресурсов по программированию в среде C++Builder являются Web-узлы, содержащие технические статьи, рецепты типа “как сделать так, чтобы...”, ответы на часто задаваемые вопросы, компоненты многократного использования и средства сторонних производителей.

Существует ряд кольцевых Web-структур, посвященных работе с C++Builder. Web-кольца позволяют посещать Web-узлы по списку, который подготовлен с учетом конкретной темы. Узлы, входящие в состав Web-кольца, содержат ссылки перехода к остальным членам этого кольца (они обычно приводятся в низу Web-страницы, чтобы можно легко переходить на другие Web-узлы этого кольца). Web-кольца, посвященные работе с C++Builder, перечислены в следующем списке.

- *Borland C++Builder Webring*. Более 40 Web-узлов.

- *The C++Builder Programmer's Ring*. Более 20 Web-узлов.
- *The Delphi Corner*. Более 70 Web-узлов.
- *Delphi Developers Webring*. Более 10 Web-узлов.

Чтобы узнать, как посетить эти Web-кольца, обратитесь к Web-узлу компании Borland по адресу: http://webring.yahoo.com/.

Рецепты типа „как сделать так, чтобы...“ и технические статьи

Наряду с Web-хранилищами компонентов, существует большое количество Web-узлов, содержащих рецепты, технические статьи и ответы на часто задаваемые вопросы по разным аспектам работы с C++Builder. Некоторые из них перечислены в следующем списке.

- *Dr.Bob's C++Builder Gate* (http://www.drbob42.com/cbuilder). Боб Свот (Bob Swart) и Рууд Ф. Пелс (Ruurd F. Pels), которые оба являются авторами этой книги, создали Web-узел C++Builder Gate. Возможно, это один из наиболее известных независимых Web-узлов, посвященных C++Builder, который регулярно обновляется за счет новых полезных статей.
- *The Bits C++Builder Information and Tutorials Site* (http://www.thebits.org). Это один из лучших узлов для получения консультаций по C++Builder.
- *BCBDEV.COM* (http://www.bcbdev.com). Этот узел содержит много прекрасных статей и ответов на часто задаваемые вопросы.
- *Bytamin-C The C++ Developers Site!* (http://www.bytamin-c.com). Сей "витамин С" содержит отличные статьи, рецепты, советы и трюки, обзоры и компоненты. Пример узла-многостаночника.
- *BCB–CAQ* (http://bcbcaq.freeservers.com). Этот CAQ-узел (Commonly Asked Questions — обычно задаваемые вопросы) содержит информацию о различных компонентах C++Builder и другие связанные с программированием статьи. Он поддерживается Дэймоном Чандлером (Damon Chandler), одним из авторов этой книги.
- *Just Another Web Site about C++ Builder* (http://www.leunen.com/cbuilder). Этот Web-узел предлагает вашему вниманию ряд полезных советов и статей.

Хранилища компонентов

Существует множество Web-узлов (их называют хранилищами компонентов), содержащих очень полезные компоненты, которые разработчики могут использовать в приложениях, сэкономив таким образом свое время (и деньги) и избавив себя от необходимости выполнять такую разработку самому. Одни компоненты распространяются бесплатно, другие продаются как условно-бесплатные или коммерческие продукты.

Поскольку C++Builder может компилировать и использовать код Delphi, у разработчиков C++Builder есть большой выбор компонентов сторонних производителей. Ниже приводится список только некоторых Web-узлов, содержащих компоненты и программные средства, которые могут вам пригодиться.

- *Codecorner* (http://www.codecorner.com). CodeCorner предлагает более 240 компонентов, разделенных по категориям.

- *ComponentSource* (http://www.componentsource.com). ComponentSource содержит тысячи компонентов для всех сред разработки, включая Delphi и C++Builder, а также компоненты ActiveX.
- *Delphi Pages* (http://www.delphipages.com). Этот узел —"полная чаша" компонентов и ресурсов для C++Builder и Delphi (более 1 500 компонентов в различных категориях).
- *DelphiSource* (http://www.delphisource.com). DelphiSource содержит большой архив компонентов для C++Builder и Delphi во многих категориях.
- *Delphi Super Page* (http://delphi.icm.edu.pl/). Здесь вы найдете компоненты для всех версий C++Builder и Delphi. Этот узел содержит более 5 500 файлов для загрузки и привлекает приблизительно 30 000 посетителей в неделю.
- *Indy* (прежде именовался *Winshoes*) *Free Internet Components* (http://www.nevrona.com/Indy). Indy — это набор бесплатных компонентов Internet, поставляемых в полном "боекомплекте".
- *Internet Component Suite* (http://users.swing.be/francois.piette/indexuk.htm). Internet Component Suite предлагает свободно распространяемый набор компонентов, предназначенных для добавления в ваши приложения средств поддержки HTTP, FTP, NMTP и других Internet-протоколов.
- *Raize Software Solutions, Inc.* (http://www.raize.com). Содержит более 90 визуальных компонентов и средство отладки CodeSite.
- *RX Library* (http://www.rxlib.com). Хранилище компонентов RxLib, содержащее множество визуальных компонентов и компонентов работы с данными, а также готовых к употреблению объектов и процедур.
- *TurboPower Software Company* (http://www.turbopower.com). TurboPower — известная своими наградами фирма-создатель наборов компонентов и программных средств. Вот некоторые из продуктов этой фирмы: Abbrevia, набор компонентов сжатия данных; AsyncProfessional для последовательных, FTP-, Fax- и постраничных коммуникаций; Internet Professional для Internet-коммуникаций; LockBox для кодирования данных; Orpheus, коллекция из более 100 специализированных компонентов; SysTools.
- *VCL Crawler* (http://www.vclcrawler.com). Относительно новый узел. На момент написания этой книги у него было более 100 компонентов для загрузки.
- *Woll2Woll* (http://www.woll2woll.com). Woll2Woll может похвастаться несколькими такими уже заслуженными наборами компонентов для C++Builder и Delphi, как InfoPower (набор значительно усовершенствованных компонентов для работы с данными, включающими элементы управления сеткой и справочной таблицей), и многими другими компонентами.

Книги и журналы

Возможно, самым популярным источником информации по разработке программных продуктов являются книги и журналы. Авторы книг имеют тенденцию вложить в свои детища максимум информации, а журналы обеспечивают постоянный поток статей профессионального уровня по актуальным проблемам. Как книги, так и журналы стараются охватить широкий диапазон тем.

В следующих разделах мы сделаем краткий обзор книг и журналов, посвященных C++Builder. Более подробную информацию о книгах, возможности их покупки в Internet-магазинах и отзывах читателей можно найти в таких заслуживающих уважения Internet-магазинах, как *Amazon.com* (`http://www.amazon.com`), *FatBrain* (`http://www.fatbrain.com`) или *Barnes and Noble* (`http://www.bn.com/`).

Выдержки из книг, публикуемые в выходных данных Macmillan USA (изд-ва Sams, Que и пр.), можно найти на узле InformIT. Выдержки чаще всего включают выбранные главы, а в некоторых случаях даже целые книги! Используйте такую возможность и читайте книги, посетив упомянутый Web-узел по адресу: `http://www.informit.com`.

Книги по теме C++Builder

До настоящего времени написано около десяти книг, посвященных C++Builder. Авторами двигало желание создать новый тип справочной книги (дополнительно к относительно короткому списку уже опубликованных справочных материалов) по C++Builder, содержащей материал, который нельзя найти ни в каких других источниках.

Ниже приводится три самых лучших книги по C++Builder с кратким описанием содержимого.

- K. Reisdorph, *Borland C++Builder 4 Unleashed,* Sams, 1999, 1248 страниц, ISBN 0672315106. Книга охватывает множество тем, предназначена для широкого диапазона читателей: от начинающих до профессионалов, более всего ориентирована на читателей среднего уровня подготовки. Описание баз данных и программирования распределенных систем составляет около 30 процентов от всего содержимого книги. *Borland C++Builder 4 Unleashed* — идеальный спутник справочника разработчика *C++Builder 5 Developer's Guide*.
- J. Miano, T. Cabanski, H. Howe, *C++Builder How-To: The Definitive C++Builder Problem Solver*, Waite Group, 1997, 822 страниц, ISBN 157169109X. Несколько устаревшая, эта книга по-прежнему содержит много полезных рецептов и методов для программистов, работающих в среде C++Builder. К сожалению, она описывает средства Borland C++Builder 3, многие из которых уже изменились в версиях 4 и 5.
- K. Reisdorph & B. Gill, *Sams Teach Yourself Borland C++Builder 4 in 24 Hours*, Sams, 1999, 451 страниц, ISBN 0672316269. Предназначенная для программистов начального и среднего уровня, эта книга содержит необходимые сведения о том, как программировать с использованием C++Builder. В ней предлагается обзор самых общих задач и методов программирования.

Благодаря сходству C++Builder и Delphi, книги, посвященные Delphi, также стоит принять во внимание. Единственное существенное различие между C++Builder и Delphi — сам язык программирования. Если вы писали на языке C++, вам не составит труда разобраться в программах на Object Pascal. Раздел "Migration from Delphi" главы 1, "Введение в C++Builder", поможет понять различия между C++ и Object Pascal.

Для получения информации о других уже изданных книгах о C++Builder воспользуйтесь поисковыми средствами в упомянутых выше Internet-магазинах, используя такие аргументы поиска, как *C++Builder*, *C++Builder* и *Delphi*. Полную копию оригинального издания книг *Charlie Calvert's Borland C++Builder Unleashed* и *Sams Teach Yourself Delphi 4 in 21 Days*, а также выдержки из книги *Sams Teach Yourself Borland C++Builder 4 in 24 Hours* можно прочитать на узле InformIt.

Книги по C++

Книг, посвященных C++, невероятно много. Назовем лишь самые популярные (они же, возможно, и лучшие).

- B. Eckel, *Thinking in C++,* Том 1 (2-е издание), Prentice-Hall, 2000, 720 страниц, ISBN 0139798099. В этом 2-ом издании его автор Брюс Икель разъясняет все детали языка программирования C++ в ясной и лаконичной манере, которая легко воспринимается как новичками, так и опытными программистами. Книга бесплатно распространяется на компакт-диске, который содержит *C++Builder Professional and Enterprise*, а также доступна для чтения по адресу: `http://www.mindview.net/ThinkingInCPP2e.html`.
- S. Meyers, *Effective C++* (2-е издание), Addison-Wesley, 1997, 304 страниц, ISBN 0201924889. Эта книга обязательно должна быть у каждого C++-разработчика. Ее автор не ставил цель преподать основы языка C++, но благодаря большому количеству примеров эта книга позволяет усовершенствовать стиль программирования на C++, особенно в области проектирования классов.
- B. Stroustrup, *The C++ Programming Language* (3-е издание), Addison-Wesley, 1997, 928 страниц, ISBN 0201889544. Создатель C++, Бьярн Страуструп, представляет полное описание языка и стандартную библиотеку шаблонов. Вряд ли эта книга подходит для начинающих, скорее ее следует рассматривать как исчерпывающий справочник по C++ и основное пособие для тех, кто всерьез решил заняться программированием на C++.

Для получения информации по другим книгам, посвященным C++, воспользуйтесь поисковыми средствами в упомянутых выше Internet-магазинах, используя аргумент поиска *C++*. Полную копию оригинального издания книги *Sams Teach Yourself C++ in 21 Days* (2-е издание), а также выдержки из других книг по C++ можно прочитать на узле InformIt.

Журналы

Тематике C++Builder уделяют внимание и журналы. Одни из них полностью посвящены C++Builder, в то время как другие содержат отдельные статьи в ряде публикаций по C++. Существует также несколько Delphi-ориентированных журналов. Приведем небольшой список, который может вам пригодиться.

- *CbuilderZine*. Прекрасно оформленный электронный журнал, посвященный C++Builder, в котором регулярно публикуются статьи консультационного плана, последние известия от Borland, обзоры новых продуктов и книг, а также предоставляется возможность загрузки компонентов. Его адрес: `http://www.cbuilderzine.com`. Кроме того, те, кто интересуется Delphi, могут обратиться к DelphiZine по адресу: `http://www.delphizine.com`.
- *C++Builder Developer's Journal*. Возглавляемый Кентом Рейсдорфом (Kent Reisdorph), автором нескольких книг по C++Builder, журнал каждый месяц публикует множество статей, написанных на профессиональном уровне ведущими разработчиками. Он распространяется через подписку. Зарегистрированные подписчики могут получать прошлые номера по электронной почте. Чтобы ознакомиться со списком прошлых выпусков, данными об условиях подписки и другой информацией, обращайтесь по адресу: `http://www.reisdorph.com`.
- *C/C++ Users Journal*. Этот ежемесячник распространяется через торговую сеть, а некоторые статьи каждый месяц можно прочитать в Internet по адресу: `http://www.cuj.com`.

Конференции и группы пользователей

Каждый год компания Borland проводит по всему миру ряд конференций (известных как BorCon). Самая большая проходила в США. Одиннадцатая ежегодная конференция проходила в июле 2000 года, на ней присутствовало более 2 200 участников. Конференции также проводятся в Австралии, Франции, Германии и Великобритании.

Эти конференции всегда имеют насыщенную программу. В течение нескольких дней специалистами компании Borland и независимыми экспертами со всего мира проводятся десятки консультаций, технических заседаний по всем продуктам Borland — включая C++Builder. Часто самое трудное — решить, какую секцию посетить! Помимо занятий по обучающей программе, проводятся открытые и закрытые слушания, выступления по отдельным продуктам, демонстрации, а также встречи в неофициальной обстановке для тех, кто "одного поля ягода", организуются специальные (это ни в коем случае нельзя пропустить) события, а для удобства участников работает компьютерная лаборатория.

Конференции — прекрасный способ интенсивного тренинга по различным темам, а также уникальная возможность тесно пообщаться с сотрудниками компании Borland, разработчиками C++Builder и его авторами. Для получения информации о предстоящих конференциях обратитесь по адресу: `http://www.borland.com/events/`.

Если вы не можете лично присутствовать на конференции, есть неплохая альтернатива — посещение групп пользователей C++Builder, Delphi или Borland. Я очень рекомендую присоединиться к одной из таких групп. В США (да и во всем мире) такие группы весьма распространены. Членские взносы таких групп обычно невелики; они расходуются на проведение консультаций, технических заседаний, а также встреч в неформальной обстановке. Это прекрасный способ узнать больше и воспользоваться чужим опытом. Для получения списка групп пользователей обращайтесь по адресу: `http://www.borland.com/programs/usergroups/uglist.html`.

Предметный указатель

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

В

Г

Д

З

И

К

Л

М

О

П

Р

С

Т

У

Ф