Михаил Краснов

bhv
www.bhv.ru
www.bhv.kiev.ua

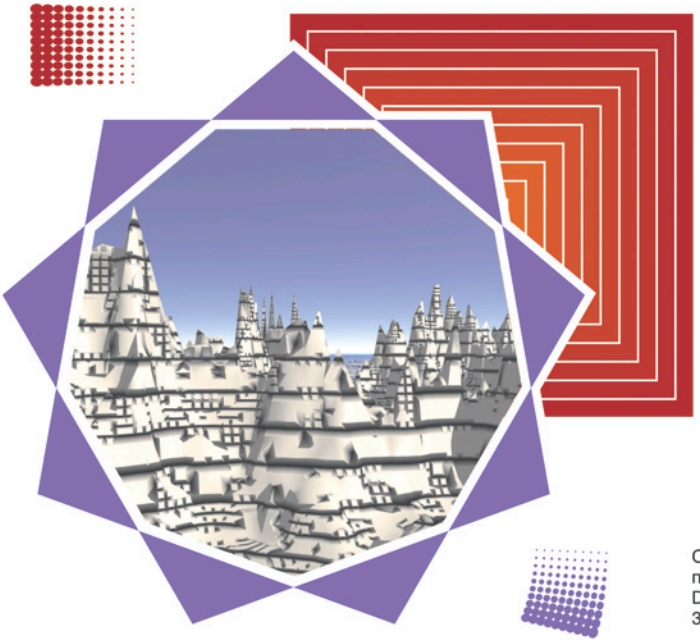
DirectX

Графика в проектах Delphi

МАСТЕР

ПРАКТИЧЕСКОЕ РУКОВОДСТВО

- Библиотеки DirectDraw и Direct3D
- Программирование игр
- Двумерная и трехмерная графика



CD-ROM содержит примеры проектов, дистрибутив ядра DirectX 8.0a и демо-версию 3D Exploration

Михаил Краснов

DirectX.

Графика в проектах *Delphi*

Дюссельдорф ◆ Киев ◆ Москва ◆ Санкт-Петербург

УДК 681.3.06

Книга посвящена использованию модулей DirectX в приложениях, разрабатываемых в Delphi. Начиная с простых примеров, последовательно и подробно рассматривается создание объектов двумерной и трехмерной графики, визуальные и цветовые эффекты, а также обсуждаются дополнительные темы, такие как быстрая работа с устройствами ввода. Большое внимание уделяется вопросам оптимизации и ускорения работы приложений. Книга содержит практические решения проблем, возникающих при программировании игр и других приложений, нуждающихся в высокой скорости вывода графики в среде Windows. Прилагается компакт-диск с инструментальными средствами, кодами и демонстрационными версиями рассматриваемых примеров.

Для широкого круга программистов

Группа подготовки издания:

Главный редактор	*Екатерина Кондукова*
Зав. редакцией	*Наталья Таркова*
Редактор	*Анна Кузьмина*
Компьютерная верстка	*Ольги Сергиенко*
Корректор	*Зинаида Дмитриева*
Дизайн обложки	*Игоря Цырульникова*
Зав. производством	*Николай Тверских*

Краснов М. В.
DirectX. Графика в проектах Delphi. — СПб.: БХВ-Петербург, 2001. — 416 с.: ил.

ISBN 5-94157-033-3

Лицензия ИД № 02429 от 24.07.00. Подписано в печать 06.06.01.
Формат 70×100 1/16. Печать офсетная. Усл. печ. л. 33,54.
Тираж 4000 экз. Заказ
"БХВ-Петербург", 198005, Санкт-Петербург, Измайловский пр., 29.

Гигиеническое заключение на продукцию, товар, № 77.99.1.953.П.950.3.99
от 01.03.1999 г. выдано Департаментом ГСЭН Минздрава России.

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

Содержание

Введение

Главной темой книги, которую вы держите в руках, является компьютерная графика, а именно использование в Delphi модулей DirectX, связанных с двумерной и трехмерной графикой.

DirectX — это набор драйверов, образующий интерфейс между программами в среде Windows и аппаратными средствами. Состоит он из набора компонентов, поддерживающих непосредственную работу с устройствами, и служит в качестве средства разработки быстродействующих мультимедийных приложений. Для программиста применение DirectX заключается в использовании набора низкоуровневых интерфейсов (API).

Развитие DirectX происходит беспрерывно и корпорация Microsoft ежегодно выпускает новую или обновленную версию этого продукта. Очередная версия включает в себя возможности предыдущих, но некоторые предлагают подходы, кардинально отличающиеся от концепций ранних версий. Так, в восьмой версии не произошло обновления модуля, связываемого с двумерной графикой, и разработчикам предложено использовать объединенный подход к графике, в котором чистая двумерная графика является частным случаем трехмерной. В этой версии единый набор API обслуживает оба подраздела компьютерной графики.

В данной книге обсуждается API седьмой и восьмой версий DirectX. В начале в ней изложено применение модуля DirectDraw для создания приложений чистой двумерной графики. DirectDraw используется как набор интерфейсов седьмой версии DirectX. Во второй части книги рассматривается компонент DirectX Graphics, как набор интерфейсов восьмой версии.

Читателю не стоит относиться к материалу книги о DirectDraw, как к устаревшему анахронизму. Во-первых, при работе с этим модулем приложения двумерной графики гарантированно используют возможности видеокарт, поддерживающих только 2D-акселерацию, а пользователей, имеющих именно такие карты, в ближайшие годы будет оставаться еще очень много.

Во-вторых, в планах разработчиков корпорации Microsoft было заявлено о том, что в девятой версии DirectX будет возобновлена поддержка DirectDraw, как обновленных интерфейсов, и этот материал наверняка будет легко "приспособить" также к новой версии. И, в-третьих, вы получите здесь представление о базовых механизмах и приемах, лежащих в основе и трехмерной графики, при переходе к которой вы встретите уже знакомые вам принципы и подходы.

Теперь я должен сказать несколько важных вещей непосредственно о книге.

Прежде всего, хочу предупредить, что книга не охватывает целиком ни DirectX, ни даже модули, относящиеся напрямую к графике. Материал чрезвычайно обширен, чтобы охватить его одним изданием, поэтому я вам не могу обещать, что вы после знакомства с моей книгой будете уметь абсолютно все, но я обещаю, что вы научитесь очень многому.

Как мне кажется, большинство читателей купят книгу в расчете на то, чтобы с ее помощью научиться создавать игры. Вы найдете в ней примеры простых игр, которые не стоит рассматривать как профессиональные. Но, познакомившись с ними, вы сможете написать и более масштабные проекты. Думаю, что книга может оказаться полезной и тем, кто не собирается программировать игры, а нуждается в средстве, позволяющем максимально быстро построить, например, диаграмму или график.

Не утверждаю, что я открыл новый жанр, но должен предупредить вас, что эта книга может показаться вам своеобразной: главный упор в ней делается на практические примеры. Среди прочитанных мною изданий по программированию самыми полезными оказались те, которые содержат не пространные рассуждения и сложные диаграммы, а те, где предлагаются готовые решения и масса примеров. Поэтому и здесь я постарался выдержать изложение в том же духе в надежде, что она принесет вам действительную пользу. В книге масса примеров, многие из которых не стоит рассматривать слишком бегло, иначе в какой-то момент вы можете потерять нить понимания. По мере неспешного ознакомления с примерами в каждом из них попробуйте что-то изменить или добавить, и тогда у вас появится ощущение полного понимания.

Эта книга и примеры к ней также очень сильно отличаются от обычных книг по программированию на Delphi. Например, здесь нет обзора компонентов, в большинстве примеров на форме располагается один компонент, а код модулей может вам поначалу показаться непривычно длинным и странным по синтаксису.

Одна из целей, которую я преследовал, состоит в том, чтобы книга читалась легко теми, кто впервые сталкивается с данной темой. Но я должен предупредить, что если вы программируете на Delphi меньше года, вам, возможно, будет очень тяжело изучать эту книгу. Вы должны хорошо знать Delphi, причем подразумевается не умение ориентироваться в палитре компонентов,

а наличие опыта в кодировании. Минимальный уровень, который вы должны иметь, чтобы приступать к чтению этой книги, таков: читатель должен свободно владеть навыками работы с невизуальными объектами, такими как объекты класса `TBitmap`. Если вы можете с помощью подобного объекта вывести на форме содержимое растрового файла, то, я надеюсь, сможете легко и быстро разобраться в материале книги.

Наверняка вы наслышаны о DirectX, и его существование не стало для вас откровением, пришедшим в вашу жизнь с этой книгой. Вы знаете, что данное средство предназначено для создания мультимедийных приложений, работающих максимально быстро, и у вас, наверное, нет вопроса ко мне, почему я написал книгу об использовании DirectX. Но, скорее всего, мне необходимо упомянуть, почему для освещения этой темы мною выбрана среда программирования Delphi. Ведь если в ней и написаны масштабные игры профессионального уровня, то их очень немного. Программисты, знающие среду Delphi поверхностно, несправедливо считают, что с ее помощью можно создавать только СУБД. А между тем, это очень мощное средство, которое годится для решения достаточно широкого круга задач. Прочитав книгу, вы убедитесь в этом. Delphi — очень популярная среда программирования, о которой разработчики DirectX позаботились не в первую очередь: для программистов, использующих C++ или Visual Basic, имеется богатый источник информации по разработке программ, комплект документации и примеров, SDK; для программистов же, использующих Delphi, таких источников информации мало. Чтобы помочь именно этой огромной армии программистов и написана данная книга. Это не руководство для тех, кто использует C++, или не умеет программировать вообще, но хочет научиться писать игры. Это учебник для тех, кто хорошо знает Delphi, но пока не умеет использовать DirectX.

Поскольку в Delphi отсутствует стандартная поддержка DirectX, нам приходится выбирать среди решений, предложенных сторонними разработчиками, главным образом, энтузиастами. Среди таких решений есть и привычное для Delphi, в виде наборов компонентов, например WDirectX и DelphiX. Но я предлагаю другое решение: мы будем использовать набор заголовочных файлов проекта JEDI. Это перенесенные энтузиастами заголовочные файлы из состава DirectX SDK корпорации Microsoft, изначально написанные на C. Такой подход хоть и приводит к кажущемуся поначалу чрезмерно громоздкому коду, но облегчит вам жизнь, когда, например, вы захотите разобраться в коде игр, написанных профессионалами. Очень многое для вас в чужом коде станет знакомым и понятным.

Обновления комплекта заголовочных файлов, а также дополнительные примеры использования DirectX в Delphi вы можете найти по ссылке **http://www.delphi-jedi.org/DelphiGraphics/**.

Заголовочные файлы, которыми я пользовался в настоящей книге, взяты мною с этого сайта, сведения об авторах трансляции указаны в коде модулей.

Здесь же вы можете найти файлы справки из состава DirectX SDK. Обратите внимание, что в файлах справки восьмой версии отсутствует информация о функциях DirectDraw, поэтому вам необходимо найти соответствующие файлы седьмой версии. Пока вы читаете первую главу книги, постарайтесь скачать эти файлы по указанному адресу.

Также вам после прочтения книги очень пригодятся переложения на Delphi примеров из DirectX SDK. Их вы найдете по адресу **http://groups.yahoo.com/group/JEDI-DirectXExamples**.

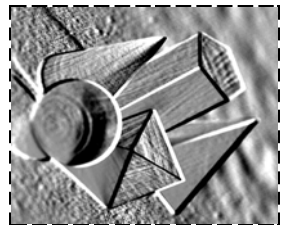
Прилагающийся к книге компакт-диск содержит инсталляцию ядра DirectX восьмой версии, но, возможно, к тому времени, когда вы начали читать эту книгу, уже появились новые версии. Их вы можете найти здесь:

- **http://www.microsoft.com/directx**
- **http://msdn.microsoft.com/directx**

По тем же адресам, наверняка, вы найдете и документацию по текущей версии.

Если у вас возникли какие-либо технические вопросы, такие, например, как проблемы с компакт-диском, обратитесь на сайт издательства **http://www.bhv.ru** (или **mail@bhv.ru**).

ГЛАВА 1



Понятие о COM

Глава является первой в серии глав, посвященных подсистеме DirectDraw. Эта библиотека, как и остальные модули DirectX, реализована в соответствии со спецификацией COM. Глава представляет собой краткий курс по этой спецификации и содержит минимальные сведения, необходимые читателю для понимания механизмов функционирования DirectDraw и других частей DirectX.

Примеры к данной главе располагаются в каталоге \Examples\Chapter01, для изучения они должны быть скопированы на диск, допускающий перезапись.

Библиотеки динамической компоновки

Ключевым понятием операционной системы Windows, позволяющим понять любую технологию, использующуюся в ней, является понятие *библиотеки динамической компоновки* (DLL, Dynamic Link Library). Любое полноценное приложение этой операционной системы (32-разрядное приложение, имеющее собственное окно) использует DLL-файлы. По мере необходимости приложение обращается к библиотекам, вызывая из них нужные функции. Например, выполнимый модуль приложения не содержит кода по отображению окна, вывода в окно и реакции на большинство событий. Перечисленные действия реализуются в системных DLL. В частности, использованием такой технологии удается экономить драгоценные ресурсы, один и тот же код не дублируется многократно, а размещается в памяти единожды.

К одной библиотеке, как правило, может обращаться одновременно несколько приложений. Библиотеку в такой схеме называют *сервером*, а обслуживаемое им приложение — *клиентом*. Сервером и клиентом в общем случае могут являться и библиотека, и приложение. В частности, это означает, что некоторая библиотека, в свою очередь, может "подгружать" функции из другой библиотеки.

Продемонстрируем работу операционной системы следующим примером. Создадим библиотеку, содержащую полезную функцию, выводящую на окне вызывающего клиента растровое изображение. Дальше приведем инструкцию ваших действий в среде Delphi. Готовый результат содержится в каталоге Ex01.

В главном меню выберите пункт **File | New** и в появившемся окне **New Items** щелкните на значке с подписью "DLL".

Чтобы выводимый растр не оказался легко доступным для посторонних глаз, скроем его, поместив в библиотеку. Для этого с помощью редактора ресурсов Image Editor (для вызова его выберите соответствующую команду меню **Tools**) создайте новый файл ресурсов с единственным ресурсом — нужным растром. Присвойте имя ресурсу — BMP1.

Для подготовки этого примера было взято одно из растровых изображений, поставляемых в составе пакета DirectX SDK, скопированное из окна редактора Microsoft Paint через буфер обмена.

Закончив редактировать растр, res-файл запишите в каталог, предназначающийся для проекта библиотеки под именем DLLRes.res.

Код DLL-проекта приведите к следующему виду:

```
library Project1;    // Проект библиотеки
uses
  Windows, Graphics;
{$R DLLRes.res}      // Подключение файла ресурсов
// Описание экспортируемой функции, размещаемой в DLL (export) и
// вызываемой стандартно (stdcall)
procedure DrawBMP (Handle : THandle); export; stdcall;
var
  wrkBitmap : TBitmap;
  wrkCanvas : TCanvas;
begin
  wrkBitmap := TBitmap.Create;
  wrkCanvas := TCanvas.Create;
  try
    // Растр загружается из ресурсов, идентифицируется именем
    wrkBitmap.LoadFromResourceName (HInstance, 'BMP1');
    wrkCanvas.Handle := Handle;
    wrkCanvas.Draw(0, 0, wrkBitmap);
  finally
    wrkCanvas.Free;
    wrkBitmap.Free;
  end;
end;
```

```
// Список экспортируемых функций
// Функция у нас единственная
exports
  DrawBMP;
// Следующий блок соответствует инициализации библиотеки
begin
end.
```

Не будет лишним привести некоторые пояснения. Аргументом функции должен являться идентификатор канвы вызываемой формы. У вспомогательного объекта процедуры, класса `TCanvas`, значение этого идентификатора устанавливается в передаваемое значение, и теперь все его методы будут работать на канве окна, вызывающего функцию приложения.

А сейчас создайте DLL, откомпилировав проект.

Внимание!

Откомпилируйте проект, но не запускайте его. Нельзя запустить DLL в понятии, привычном для обычного приложения.

В каталоге должен появиться файл Project1.dll. Исследуйте библиотеку: поставьте курсор на ее значок, нажмите правую кнопку мыши и в появившемся контекстном меню выберите команду **Быстрый просмотр**.

Примечание

Если данная команда отсутствует, вам необходимо установить соответствующий компонент, входящий в дистрибутив операционной системы.

В окне отобразится информация о содержимом библиотеки, разбитая по секциям, среди которых нас особо интересует секция экспортируемых функций (рис. 1.1).

Если с помощью утилиты быстрого просмотра вы взглянете на содержимое модуля обычного приложения, то не найдете там секции экспортируемых функций. Это принципиальное отличие библиотек динамической компоновки от обычных исполняемых файлов.

Примечание

Некоторые библиотеки скрывают секцию экспортируемых функций от обычного просмотра, но она там обязательно присутствует, даже если библиотека содержит только ресурсы. Пример подобной библиотеки — системная библиотека moricons.dll.

Итак, созданная нами библиотека содержит код экспортируемой функции с именем `DrawBMP` и растровое изображение. Сервер готов. Теперь создайте клиента. Организуйте новый проект, сохраните его в другом каталоге (готовый проект содержится в каталоге Ex02).

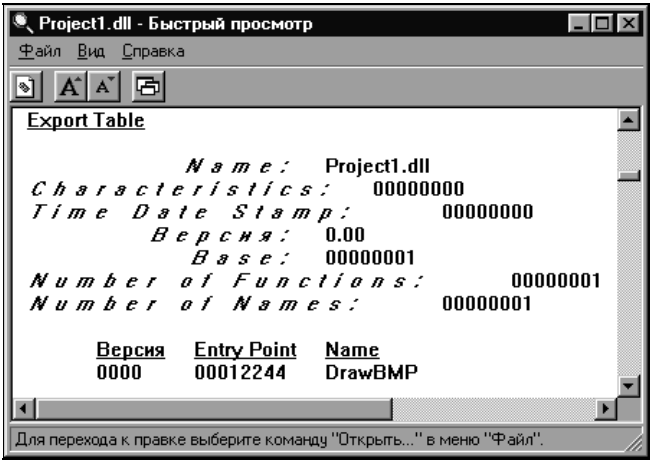


Рис. 1.1. Убеждаемся, что в секции экспортируемых функций созданной библиотеки присутствует название DrawBMP

В секции `implementation` введите следующую строку:

```
procedure DrawBMP (Handle : THandle); stdcall; external 'Project1.dll';
```

Этим мы декларируем нужную нам функцию. Ключевое слово `external` указывает, что данная функция размещена в библиотеке с указанным далее именем. Ключевое слово `stdcall` определяет вызов функции стандартным для операционной системы образом. При использовании импортируемых функций такие параметры задаются обязательно.

На форме разместите кнопку, в процедуре обработки события щелчка кнопки мыши которой введите строку:

```
DrawBMP (Canvas.Handle);
```

Аргументом вызываемой функции передаем ссылку канвы окна. Основной смысл этой величины — идентификация полотна окна.

Откомпилируйте проект, но пока не запускайте. С помощью утилиты быстрого просмотра исследуйте содержимое откомпилированного модуля: найдите в списке импортируемых функций следы того, что приложение использует функцию `DrawBMP` (рис. 1.2).

Обратите внимание, что имя известной нам функции находится в длинном ряду имен других импортируемых приложением функций. То есть модуль даже минимального приложения использует целый сонм функций, подключаемых из DLL. Именно о них упоминалось в начале главы как о функциях, ответственных за появление окна приложения и вывод на канве окна, а также отвечающих за реакцию окна на события. Эти функции именуются *системными*. Так же называются и библиотеки, хранящие код таких функций.

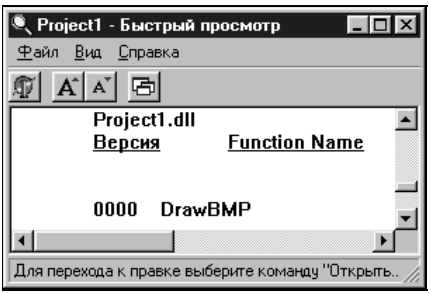


Рис. 1.2. Информация обо всех импортируемых приложением функциях доступна для анализа

Примечание

Другое название системных функций — функции API (Application Program Interface).

При исследовании содержимого созданной нами библиотеки вы могли обратить внимание, что она ко всему прочему импортирует массу функций из системных библиотек.

Delphi позволяет нам писать краткий и удобочитаемый код, но при компиляции этот код преобразуется к вызову массы системных функций, и подчас одна строка кода "расшифровывается" вызовом десятка функций API. Программирование на Delphi образно можно себе представить как общение с операционной системой посредством ловкого переводчика, способного нам одной фразой передать длинную тираду, перевести без потери смысла, но некоторые потери мы все-таки имеем. Прежде всего, мы расплачиваемся тем, что приложения, созданные в Delphi, как правило, имеют сравнительно большой размер. Другая потеря — скорость работы приложения. При использовании библиотеки VCL и концепции объектно-ориентированного программирования вообще, мы жертвуем скоростью работы приложения.

В тех случаях, когда скорость работы приложения чрезвычайно важна, как в случае с обработкой графики, выход может состоять в том, чтобы отказаться от применения "переводчика", писать код, основанный исключительно на использовании функций API. Но такие программы плохо понятны новичкам, требуют специальной подготовки, поэтому мы не будем злоупотреблять этим. Если вы испытаете необходимость подробного разговора о том, как создавать в Delphi приложения без вызова библиотеки классов VCL, то автор может посоветовать вам свою предыдущую книгу, в списке литературы она поставлена на первое место. В ней вы найдете достаточно примеров подобных проектов. Ну а в этой книге постараемся не приводить таких примеров.

Вернемся к нашему примеру. Если вы сейчас запустите приложение, то вас постигнет неудача: сразу после запуска появится системное окно с сообще-

нием о том, что необходимый файл библиотеки не найден. Ничего удивительного, но обратите внимание, что сообщение об ошибке появляется сразу же после запуска приложения, а не после вызова функции, вслед за нажатием кнопки.

Скопируйте в этот же каталог скомпилированную библиотеку и снова запустите приложение. Теперь при запуске все должно быть в порядке, никаких сообщений не появится, а после нажатия кнопки на поверхности окна должна отобразиться картинка (рис. 1.3).

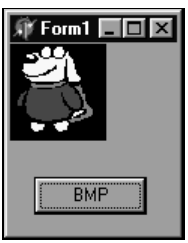


Рис. 1.3. Особенность примера состоит в том, что код для вывода картинки не содержится в модуле приложения

Главное в рассмотренном примере заключается в том, что код приложения не содержит напрямую ничего, связанного с отображаемой в окне картинкой. Приложение обращается к указанной нами библиотеке динамической компоновки, которая выполняет всю работу по выводу изображения.

Первый наш пример является моделью диалога приложения с библиотеками вообще. Каждый раз, когда нам это необходимо, работает библиотека, путем вызова нужной функции.

Посмотрим внимательнее на работу приложения. Картинка исчезает при каждой перерисовке окна, например, если минимизировать, а затем восстановить окно, то картинка "пропадет". Объяснить это легко: при перерисовке окна вызывается собственный обработчик события `OnPaint` окна, а мы позаботились о наличии в нем кода, с помощью которого можно было бы запоминать текущий вид окна. Операционная система подобную услугу не предоставляет, поскольку на нее требуется слишком много ресурсов. Шлифовать код этого примера не станем, мы получили от него почти все, что требовалось для нас.

Запустите несколько копий клиентов и протестируйте вывод картинки на поверхность каждого из них. Пример упрощенный, но я, надеюсь, он смог достичь главной цели, преследуемой мною. Использование динамических библиотек является действительно эффективной технологией построения архитектуры программных систем: код клиентов освобожден от дублирования.

Еще одно важное свойство динамических библиотек состоит в том, что при их использовании безразлично, в какой программной системе созданы клиенты и сами библиотеки. Этим мы пользуемся во время применения DirectX

в проектах Delphi точно так же, как и при использовании любой системной библиотеки.

В коде клиента указывается имя вызываемой функции, но во время работы откомпилированного приложения клиент при вызове динамической библиотеки ориентируется не по имени функции, а по соответствующей функции точке входа, адрес которой он получает при инициализации библиотеки. Взгляните снова на рис. 1.1. Слева от имени экспортируемой функции вы найдете адрес точки входа. Клиент при инициализации библиотеки получает это значение в качестве опорного для вызова функции.

Вспомним, что при запуске исполнимого модуля клиента происходит исключение при отсутствии необходимой библиотеки, рассмотренная компоновка приложения называется *статическим связыванием*.

Динамическое связывание отличается тем, что клиент загружает библиотеку не сразу же после своего размещения в памяти, т. е. запуска, а по мере надобности. Примером такого подхода является проект каталога Ex03. В разделе `implementation` модуля записано следующее:

```
type     // Процедурный тип функции, подгружаемой из библиотеки
  TDrawBMP = procedure (Handle : THandle); stdcall;
// Щелчок кнопки с надписью BMP
procedure TForm1.Button1Click(Sender: TObject);
var
  hcDll : THandle;                          // Указатель на библиотеку
  procDrawBMP : TDrawBMP;                   // Подгружаемая функция
begin
  hcDll := LoadLibrary('Project1.dll');     // Динамическая загрузка DLL
  if hcDll <= HINSTANCE_ERROR then begin    // Загрузка не удалась
     MessageDlg ('Отсутствует библиотека Project1!', mtError, [mbOK], 0);
     Exit;
  end;
  // Библиотека загружена. Получаем адрес точки входа нужной функции
  procDrawBMP := GetProcAddress(hCDll, 'DrawBMP');
  // проверка на успешность операции связывания
  if not Assigned (procDrawBMP) then begin
     MessageDlg (В библиотеке Project1.dll отсутствует нужная функция!,
                 mtError, [mbOK], 0);
     Exit;
  end;
  procDrawBMP (Canvas.Handle);    // Вызываем функцию
  FreeLibrary(hcDll);             // Выгружаем библиотеку
end;
```

Схема наших действий теперь такова: загружаем библиотеку только в момент, когда она действительно необходима, получаем адрес требуемой функции и обращаемся к ней. Обратите внимание, что успешная загрузка

библиотеки не является окончательным признаком того, что мы можем успешно использовать необходимую нам функцию. В каталог этого проекта автор поместил "испорченную" библиотеку Project1.dll, в ней отсутствует нужная нам функция.

Подобная ситуация на практике вполне возможна, если у одной и той же библиотеки есть несколько версий, различающихся набором функций. Производитель подчас распространяет несколько версий программы, различающихся функциональностью, и использующие их клиенты должны перед вызовом функций производить проверку на действительное присутствие этих функций в библиотеке.

Протестируйте работу проекта, заменив библиотеку в его каталоге "правильной", из каталога самого первого примера.

Динамическая загрузка библиотек используется знакомыми вам приложениями очень часто. Например, при проверке правописания текстовый редактор загружает соответствующую библиотеку только при установленном режиме проверки.

COM-модель

Технология, основанная на динамических библиотеках, является очень эффективной, потому и стала основой программной архитектуры операционной системы. Однако ей присуще ограничение, не позволяющее использовать парадигму объектно-ориентированного программирования (ООП): библиотеки могут содержать код функций и процедур, а также ресурсы, но не способны содержать описания классов. Это утверждение верно отчасти, я говорю пока о DLL "в чистом виде". По мере развития программирования как технологии, возникла необходимость поддержки ООП на уровне операционной системы.

Самым ходовым примером такого использования идей ООП на уровне операционной являются составные документы. Вставляя в текстовый документ электронную таблицу или записывая в нем математическую формулу с помощью редактора формул, пользователь текстового процессора как раз встречается со зримым воплощением ООП. Вставленный, внедренный документ является объектом со своими свойствами и методами. Это пример зримого воплощения технологии COM (Component Object Model, модель компонентных объектов). Хотя я и упомянул в примере составные документы, COM предоставляет концепцию взаимодействия программ любых типов: библиотек, приложений, системного программного обеспечения и др. Для нашей темы важно подчеркнуть, что COM стала частью технологий, не имеющих никакого отношения к составным документам.

COM может применяться для создания программ любых типов, в частности DirectX использует эту технологию. Поэтому мы и вынуждены сделать небольшой экскурс в эту тему.

Первоначально для всей группы технологий, в основе которых лежит COM, корпорацией Microsoft было предложено общее имя — OLE. Затем, по мере развития и дополнения технологии, это название менялось. Например, однажды оно стало ActiveX, но программисты со стажем часто так и продолжают пользоваться термином OLE (сейчас это не является аббревиатурой) для обозначения данной группы технологий.

COM — не язык, не протокол. Это метод взаимодействия между программами и способ создания программ.

Функции программы, доступные для использования другим программам, называются *сервисами*. COM определяет стандартный механизм, с помощью которого одна часть программного обеспечения предоставляет свои сервисы другой.

Для нас особенно важно то, что технология COM также является независимой от языка программирования. Физически приложение, предоставляющее сервисы, может быть реализовано в виде обычного выполнимого модуля, либо, чаще всего, реализовано в виде библиотеки. Как и в случае обычных библиотек, неважно, в какой программной системе созданы серверы и использующие их клиенты. В случае с обычной DLL-библиотекой клиенту достаточно знать адрес точки входа нужной функции и в определенный момент передать управление по этому адресу. Тот факт, что библиотека должна предоставлять не обычные функции, а методы объектов, внес в эту схему некоторые изменения, о которых мы поговорим позже.

Контроль версии

Сервером может быть целый программный комплекс, а не один-единственный файл.

При распространении библиотеки ее обычно помещают в какой-либо общедоступный каталог, например системный. Это вам знакомо, поскольку встречалось при установке программ. Наверняка вам известны и возникающие при этом проблемы. Например, при самостоятельном удалении таких программ сопутствующие библиотеки могут остаться, хотя больше никто их не использует.

Если же сервер представляет собой не один файл, а внушительный набор модулей, то размещение его целиком в системном каталоге принесло бы массу дополнительных проблем пользователю, который не сможет правильно определить назначение каждого файла из десятка установленных в системном каталоге.

Перед разработчиками операционной системы стояла следующая задача: необходимо предоставить клиенту возможность доступа к серверу независимо от места его физического расположения. Пусть пользователь устанавли-

вает программы там, где это ему необходимо, хоть и не в общедоступном каталоге, а клиентские программы должны получать доступ к серверу, где бы он ни располагался.

Один из способов решения задачи таков: при установке программы в файл автозагрузки дописывается строка, объявляющая каталог устанавливаемой программы доступным для всех приложений. Теперь при каждом поиске файла система будет заглядывать и в этот каталог. Подобное решение малоэффективно и удовлетворительным являлось лишь два десятилетия назад, когда на одном компьютере установить больше десятка крупных программ практически было невозможно. Сегодня же на компьютере пользователя могут быть установлены одновременно сотни приложений, и блуждание по каталогам может оказаться чересчур долгим. К тому же библиотеки разных производителей, с различным набором функций, могут быть случайно названы одинаково, и клиенту первым может попасться не тот сервер, который он ищет.

Итак, клиент в любой момент должен иметь точную информацию о текущем расположении нужного ему сейчас сервера.

Найденное разработчиками решение состоит в использовании базы данных установленных программ. Такая база данных носит название *реестр*. Функции ее гораздо шире названной мною, но я сосредоточусь только на ней. При установке сервер записывает в реестр свой уникальный идентификатор и, как минимум, информацию о собственном физическом расположении. Клиент при вызове сервера обращается к базе данных установленных программ, ориентируясь по идентификатору, находит и загружает либо запускает сервер. Клиенту, в принципе, можно и не знать имени необходимой библиотеки, главное должен быть известен ее идентификатор. Схема взаимодействия клиента и сервера мною упрощена, напрямую они не общаются, но, надеюсь, основное я сумел донести.

Глобальный вопрос, мучающий впервые прикоснувшихся к этой теме, можно сформулировать так: "Почему это здесь?". DirectX является частью операционной системы, он неизбежно присутствует в ней сразу же после установки. Хоть он и реализован в виде набора файлов, но помещаются они всегда в системный каталог, и ничего зазорного в этом для системных файлов нет. Первый, но не самый главный, ответ на этот вопрос вы уже получили: разработчики стремились отразить требование сегодняшнего дня, связанное с поддержкой ООП на уровне операционной системы. Приступая к разработке DirectX, разработчики корпорации Microsoft задались целью создать набор объектно-ориентированных библиотек, и COM-модель подходит здесь как нельзя лучше.

Подчеркну, что данная книга не является официальным документом, все, что вы в ней читаете, является мыслями автора, и не более. Многие мысли основаны на официальных документах, но далеко не все.

Например, я твердо убежден, что DirectX можно было бы и не строить на основе COM-модели. Для обеспечения его функциональности технологии использования "обычных" библиотек вполне достаточно, а для графической части системы ООП является подспорьем незначительным. Тем более что в технологии COM имеются ограничения с точки зрения традиционного ООП, а новичкам изучение COM часто тяжело дается. Нередко для наглядности при изучении парадигмы ООП прибегают к визуальным иллюстрациям, но сама техника программирования компьютерной графики очень хорошо описывается и стародавним процедурным подходом.

Итак, если бы DirectX не был основан на COM, он в чем-то, может быть, и выиграл. Но это не значит, что весомых оснований в решении разработчиков построить DirectX именно на основе COM-технологии нет.

Существенное преимущество COM-серверов перед обычными библиотеками состоит в облегчении контроля версии сервера. С самого начала работы над DirectX его разработчики были убеждены в том, что одной версией они не ограничатся, и каждая последующая версия продукта будет снабжена новыми, дополнительными функциями. А некоторые прежние функции будут изменяться, например, в связи с устранением ошибок.

В случае с традиционными DLL каждое новое обновление продукта порождает у разработчиков массу проблем. Можно новые функции располагать в библиотеках с новым именем, а старые функции клиентами будут загружаться из прежних библиотек. Это плохо, поскольку влечет потери времени.

Если же новая версия сервера реализована физически в файлах с прежним названием, как серверу узнать, запрашивает ли клиент старую версию функции или новую? Ведь наряду с клиентами, появившимися после выхода новой версии сервера, его будут использовать и клиенты, созданные до появления новой версии. Эти клиенты ничего не знают о новых функциях и изменениях в реализации функций, носящих прежнее имя. Конечно, в библиотеку можно поместить информацию о версии продукта, но тогда в коде каждой функции надо хранить информацию о том, к какой версии сервера она относится. Если добавляется очень много функций, то все это выливается в массу проблем для разработчиков сервера и клиентов.

Вдобавок остается проблема с беспорядочным размещением файлов библиотек на диске: одни и те же файлы могут многократно копироваться на жестком диске в разные каталоги. Или поверх обновленной версии может быть установлена более старая.

Технология COM тем и отличается от традиционных библиотек, что хорошо приспособлена к решению проблемы контроля версии сервера. Хочу подчеркнуть, что все эти проблемы устранимы и в схеме традиционных DLL, но решения получаются громоздкими и способны привести к ошибкам. С использованием же технологии COM появляется гарантия, что сервер не будет установлен многократно, а клиент станет получать именно запрашиваемый набор функций.

COM-объекты

Как уже отмечалось, технология COM появилась вслед за возникшей потребностью программистов получить реализацию парадигмы ООП. В COM любая часть программного обеспечения реализует свои сервисы как один или несколько объектов COM.

COM-объекты представляют собой двоичные программные компоненты, подобно компонентам Delphi, устанавливаемым на уровне операционной системы и доступным для использования в любой среде программирования. COM-объекты для Object Pascal ничем, по сути, не отличаются от обычных объектов, или, по крайней мере, очень похожи на обычные невизуальные объекты, такие как объекты класса `TBitmap`. Изучение DirectX позволит нам разобраться с методами невизуальных объектов особых типов. Только необходимо сразу же запомнить, что у COM-объектов нет свойств, есть только методы. Вдобавок, коренное отличие таких объектов состоит в использовании конструкторов и деструкторов.

Для создания COM-объекта не вызывается функция конструктора, как для обычных объектов в Delphi. Первым нашим действием будет создание *главного объекта*, который имеет методы, использующиеся для создания других объектов и получения необходимых интерфейсов.

Для удаления COM-объекта вместо метода `Free` обычно предназначен метод `_Release`. Это справедливо в общем случае, но иногда для освобождения памяти, занятой COM-объектом, будем просто присваивать значение `nil` соответствующей переменной.

Интерфейсы

Интерфейсом обозначается набор функций, предоставляемый некоторым предложением. Обычные приложения предоставляют один интерфейс, т. е. весь тот набор функций, который реализован в вашей, к примеру, бухгалтерской программе, является в такой терминологии единым интерфейсом. Если бы ваша бухгалтерская программа могла предоставлять несколько наборов функций, то она имела бы несколько интерфейсов.

Здесь начинающие обычно испытывают затруднение. Вопрос, зачем же DirectX предоставляет несколько интерфейсов, кажется резонным.

Вспомним еще раз проблему контроля версии. Клиент может запрашивать функцию или набор функций, реализованных в различных версиях DirectX, по-разному. Очень важно предоставлять ему эти функции именно в той реализации, как он того ожидает. Например, если в какой-либо предыдущей версии функция реализована с известной ошибкой, то клиент при использовании этой функции может делать поправку на данную ошибку. Тогда,

если клиент получит уже скорректированную функцию, такая поправка может только испортить все дело.

DirectX предоставляет несколько интерфейсов, связанных с различными версиями. Клиент запрашивает именно тот интерфейс, который ему известен. Например, клиент создан пару лет назад и просто ничего не знает о новых функциях, появившихся в DirectX с тех пор. Функции, чьи имена не изменились, но реализация в последующих версиях сервера претерпела изменения, должны работать именно так, как они работали во времена создания клиента, и как того ожидает клиент.

Конечно, подобная схема отнюдь не идеальна. Например, если функция в новой версии реализована эффективнее, то "старый" клиент просто не сможет ею воспользоваться, он запустит ее старую версию. Поэтому при установке новой версии DirectX не приходится ожидать, что ранее установленные игры автоматически станут выглядеть иначе. Но все же это одно из самых эффективных решений.

Итак, сервер поддерживает один или несколько интерфейсов, состоящих из методов. Клиенты могут получить доступ к сервисам только через вызовы методов интерфейсов объекта. Иного непосредственного доступа к данным объекта у них нет.

Все COM-интерфейсы унаследованы от интерфейса, называемого `IUnknown`, обладающего тремя методами: `QueryInterface`, `AddRef` и `_Release`. О них нам надо знать совсем немного, ведь непосредственно к графике они отношения не имеют.

Последний в этом списке метод мы уже вскользь обсуждали — удаление объекта. Часто использование его будем заменять простым освобождением памяти.

Предпоследний метод предназначен для подсчета ссылок на интерфейсы. Клиент явно инициирует начало работы экземпляра COM-объекта, а для завершения его работы он вызывает метод `_Release`. Объект ведет подсчет клиентов, использующих его, и когда количество клиентов становится равным нулю, т. е. когда счетчик ссылок становится нулевым, объект уничтожает себя сам. Новичок может здесь растеряться, поэтому я уточню, что мы всем этим не будем пользоваться часто, и вы можете особо не напрягать внимание, если все это кажется сложным. Просто клиент, получив указатели на интерфейсы объекта, способен передать один из них другому клиенту, без ведома сервера. В такой ситуации ни один из клиентов не может закончить работу объекта с гарантией того, что делает это преждевременно. Пара методов `AddRef` и `_Release` дает гарантию того, что объект исчезнет только тогда, когда никто его не использует.

Обычно свой первый указатель на интерфейс объекта клиент приобретает при создании главного объекта. Имея первый указатель, клиент получает

указатели на другие интерфейсы объекта, методы которых ему необходимо вызывать, запрашивая у объекта эти указатели с помощью метода `QueryInterface`.

Перейдем к иллюстрации. В этом проекте мы должны сообщить пользователю, возможно ли применять на данном компьютере DirectX седьмой версии. Это самое простое приложение, использующее DirectDraw, и здесь нет графики, мы только определяемся, возможна ли в принципе дальнейшая работа. У DirectDraw нет интерфейсов восьмой версии, и наше приложение не различит седьмую и последующие версии. Позже мы сможем распознать присутствие именно восьмой версии, а пока что наши приложения пусть довольствуются и предыдущей.

Можете взять готовый проект из каталога Ex04, но будет лучше, если вы повторите все необходимые действия сами.

Создайте новый проект и в его опции **Search path** запишите путь к каталогу, содержащему заголовочный файл DirectDraw.pas, в моем примере там записано "..\..\DUnits".

В разделе `private` опишите две переменные:

```
FDD : IDirectDraw;
FDD7 : IDirectDraw7;
```

Первая из них является главным объектом DirectDraw. Вторая переменная нужна, чтобы проиллюстрировать применение метода `QueryInterface`. В используемом нами сейчас модуле `DirectDraw.pas` найдите строки, раскрывающие смысл новых для нас типов:

```
IDirectDraw = interface;
DirectDraw7 = interface;
```

Ключевое слово `interface` здесь, конечно, является не началом секции модуля, а типом, соответствующим интерфейсам COM-объектов.

Обработчик создания окна приведите к следующему виду:

```
procedure TForm1.FormCreate(Sender: TObject);
var
    hRet    : HRESULT;     // Вспомогательная переменная
begin
    // Создание главного объекта DirectDraw
    hRet := DirectDrawCreate (nil, FDD, nil);
    if Failed (hRet)      // Проверка успешности предыдущего действия
       then ShowMessage ('Ошибка при выполнении DirectDrawCreate')
       // Поддерживается ли интерфейс 7-й версии DirectX
       else hRet := FDD.QueryInterface (IID_IDirectDraw7, FDD7);
    if Failed (hRet)        // Или один из двух,
                            // или оба интерфейса не получены
```

```
     then ShowMessage ('DirectX 7-й версии не доступен')
     else ShowMessage ('DirectX 7-й версии доступен');
    // Освобождение памяти, занятой объектами
    if Assigned (FDD7) then FDD7 := nil;
    if Assigned (FDD) then FDD := nil;
end;
```

Уже при подготовке этого, простейшего, примера я прибегнул к некоторым упрощениям, но все равно у каждого новичка здесь появится масса вопросов. Попробую предвосхитить и разрешить их.

Итак, первая строка кода — создание главного объекта, через интерфейсы которого выполняются действия по созданию остальных объектов. Как я говорил, для COM-объектов нельзя использовать обычный конструктор.

Переменная `DirectDrawCreate` описывается в заголовочном файле DirectDraw.pas так:

```
DirectDrawCreate : function (lpGUID: PGUID;
                             out lplpDD: IDirectDraw;
                             pUnkOuter: IUnknown) : HResult; stdcall;
```

При инициализации модуля происходит связывание переменной и получение адреса точки входа:

```
DirectDrawCreate := GetProcAddress(DDrawDLL,'DirectDrawCreate');
```

Это нам немного знакомо по первому примеру. Здесь происходит динамическая загрузка функции из библиотеки. Ссылка на библиотеку описывается так:

```
var
  DDrawDLL : HMODULE = 0;
```

Первое действие при инициализации модуля — загрузка библиотеки:

```
DDrawDLL := LoadLibrary('DDraw.dll');
```

С помощью утилиты быстрого просмотра можем убедиться, что действительно в списке экспортируемых функций данной библиотеки (обратите внимание, что этот список сравнительно невелик) присутствует имя функции `DirectDrawCreate`. Напоминаю, что сам файл библиотеки содержится в системном каталоге, как правило, это C:\Windows\System\. Оттуда загружается функция. Но каков смысл ее аргументов и возвращаемой ею величины? Начнем с возвращаемой величины. Из описания ясно, что тип ее — `HRESULT`, который имеет результат всех функций, связанных с OLE. Обрабатывается результат таких функций для проверки успешности каких-либо действий, как в данном случае, для того, чтобы выяснить, успешно ли выполнена операция получения интерфейса.

Это 32-битное целое значение, описание типа которого вы можете найти в модуле system.pas:

```
HRESULT = type Longint;
```

HRESULT — общий для OLE тип, соответствующий коду ошибки. Каждый сервер по-своему распределяет возможные ошибки и возвращаемый код. Общим является то, что нулевое значение эквивалентно отсутствию ошибки.

Коды ошибок, возвращаемых функциями, связанными с DirectDraw, можно интерпретировать в осмысленную фразу с помощью функции

```
function DDErrorString(Value: HResult) : string;
```

Эта функция описана в модуле DirectDraw.pas. Аргументом ее является код ошибки, результатом — строка, раскрывающая смысл произошедшей неудачи. Равенство нулю кода выступает признаком успешно выполненной операции. Анализ успешности операции часто выполняется просто сравнением возвращаемой величины с константой DD_OK, равной нулю.

Примечание

Константа S_OK, равная нулю, также может применяться во всех модулях, использующих OLE, но обычно каждый из них определяет собственную нулевую константу.

В примере для оценки успешности операции я пользуюсь системной функцией, описанной в модуле windows.pas:

```
function Failed(Status: HRESULT): BOOL;
```

Функция возвращает значение True, если аргумент отличен от нуля. Есть и обратная ей функция, возвращающая значение True при отсутствии ошибок:

```
function Succeeded(Status: HRESULT): BOOL;
```

Теперь вернемся к аргументам функции DirectDrawCreate. Первый из них задает параметры работы приложения, если задавать значение его в nil, то при работе будет применяться текущий видеодрайвер. Если же необходимо строго оговорить, чтобы приложение не использовало все преимущества аппаратного ускорения, то это значение нужно установить так:

```
PGUID(DDCREATE_EMULATIONONLY)
```

Если же требуется оговорить, что создаваемый объект DirectDraw не будет эмулировать особенности, не поддерживаемые аппаратно, надо использовать в качестве этого параметра константу DDCREATE_HARDWAREONLY. Функция DirectDrawCreate тогда "проглотит" аргумент в любом случае, но, в будущем, попытка вызвать методы, требующие неподдерживаемые особенности, приведет к генерации ошибки с кодом DDERR_UNSUPPORTED.

Второй параметр функции — собственно наш объект, который примет данные.

Последним аргументом всегда надо указывать `nil`. Этот параметр зарезервирован для будущих нужд, чтобы старые приложения смогли в перспективе работать при измененной COM-модели.

Так, все, связанное с первым действием, — созданием главного объекта, — разобрали. Функция `DirectDrawCreate` вряд ли когда-либо возвратит ненулевое значение. Это будет соответствовать ситуации серьезного сбоя в работе системы. Однако после каждого действия необходимо проверять успешность его выполнения. Приходится сразу же привыкнуть к тому, что код будет испещрен подобной проверкой, анализом возвращаемого функцией значения. Некоторые действия вполне безболезненно можно выполнять без проверки на неудачу, поскольку ошибки при их выполнении если и возможны, то крайне редки. Ключевые же операции следует обязательно снабжать подобным кодом, поскольку ошибки при их выполнении вполне возможны и даже ожидаемы. Появляются эти исключения не по причине неустойчивого поведения системы или приложения, а закономерно в ответ на изменения окружения работы приложения. Например, пользователь может временно переключиться на другое приложение или поменять параметры рабочего стола по ходу работы вашего приложения. Анализ исключений позволяет вашему приложению отслеживать такие моменты и реагировать на изменившиеся условия работы.

Дальше в коде примера идет следующая строка:

```
FDD.QueryInterface (IID_IDirectDraw7, FDD7);
```

Вызываем метод `QueryInterface` главного объекта для получения нужного нам интерфейса, соответствующего седьмой версии DirectX. Заглянем в описание этого интерфейса. Начало выглядит так:

```
IDirectDraw7 = interface (IUnknown)
  ['{15e65ec0-3b9c-11d2-b92f-00609797ea5b}']
```

Все интерфейсы строятся на базе интерфейса `IUnknown`, в следующей строке указывается идентификатор конкретного интерфейса, за которой приведено перечисление его методов. Идентификаторы интерфейсов используются при взаимодействии клиента с сервером. Следы наиболее важных идентификаторов мы можем обнаружить в реестре. Например, в заголовочном файле вы можете найти такие строки описания идентификаторов:

```
const
  CLSID_DirectDraw: TGUID = '{D7B70EE0-4340-11CF-B063-0020AFC2CD35}';
  CLSID_DirectDraw7: TGUID = '{3c305196-50db-11d3-9cfe-00c04fd930c5}';
```

Запустив системную программу редактирования реестра regedit.exe и активизировав поиск любого из этих идентификаторов, вы способны найти соответствующие записи в базе данных (рис. 1.4).

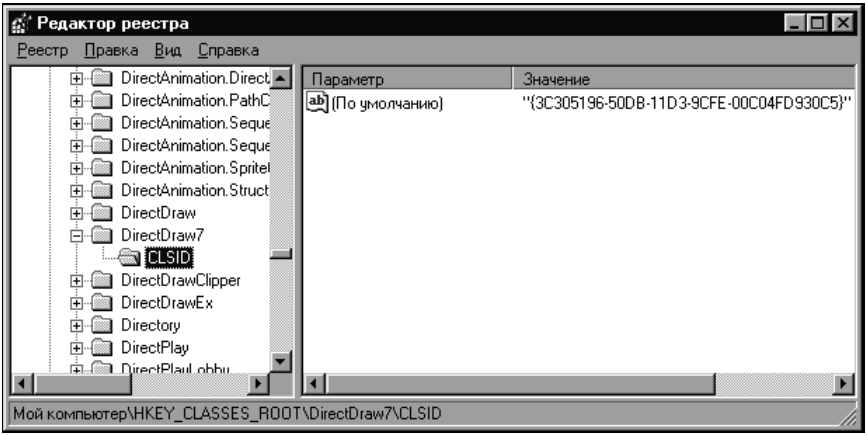


Рис. 1.4. По значению идентификатора интерфейса клиент находит запись о сервере в реестре

Я изрядно упрощаю рассмотрение тонких вопросов, связанных с COM-моделью, но для успешного использования DirectX нам таких общих представлений о ней будет вполне достаточно.

Аргументов у метода `QueryInterface` два: запрашиваемый интерфейс и объект, в который должен помещаться результат.

Дальше в нашей программе идет проверка успешности предыдущего действия, по традиционной схеме. Обратите внимание, что другой признак провала конкретно этой операции заключается в том, что значение `FDD7` окажется равным `nil`. COM-объекты в этом плане для нас будут такими же, как и обычные объекты в Delphi, признаком связанности объектов является наличие каких-либо данных в них.

Попутно еще одно важное замечание. В начале работы необходимо установить в `nil` значение всех переменных, соответствующих COM-объектам. Только из желания упростить код я не сделал этого в программе, но в последующих примерах будем строго следить за выполнением данного правила. Все подобные мероприятия кажутся необязательными, но невыполнение их только повышает вероятность некорректной работы вашего приложения.

Тот факт, что нам не удастся получить указатель нужного интерфейса, является вполне возможным, например, у пользователя просто не установлен DirectX необходимой нам версии. Клиент запрашивает интерфейс седьмой версии, и получит его именно в таком виде, как он того ожидает, даже если установлен DirectX старшей версии.

После информирования пользователя о том, установлен ли у него DirectX нужной нам версии, работа программы завершается, и память, занятая COM-объектами, освобождается. Последнее действие тоже является процедурой, обязательной для всех наших примеров. Если этого не делать, то

приложение может при выходе порождать исключения. Другая возможная ситуация: приложение корректно работает при первом запуске, а после его закрытия ни то же самое приложение, ни любое другое, использующее DirectX, корректно работать уже не может. Каждый раз, когда вы встречаетесь с подобной ситуацией, помните, что вина за это целиком лежит на вашем приложении. Такие простые программы, как разбираемая нами сейчас, навряд ли приведут к похожим авариям, но будем привыкать делать все правильно.

Память, занятую объектами, мы освобождаем в порядке, обратном порядку их связывания. Данное правило тоже очень важно соблюдать. Использование функции `Assigned` вполне можно заменить сравнением значения переменной с `nil`, в этом плане все выглядит также обычно, как и при работе с самыми заурядными объектами Delphi.

Из всех предопределенных методов интерфейсов метод `QueryInterface` является самым важным. Но и им мы, в дальнейших примерах, пользоваться не будем.

Рассматриваемый пример может подсказать нам, какие действия надо предпринимать в распространяемых приложениях, чтобы они корректно работали в ситуации отсутствия на пользовательском компьютере нужной нам версии DirectX. Но в остальных примерах инициализацию DirectDraw подобным образом проводить не будем, подразумевая, что нужные интерфейсы присутствуют.

Важное замечание: рассмотренный порядок действий в начале работы приложения является самым надежным для случаев, если приложение может быть запущено на компьютерах, не располагающих DirectX версии 7 и выше. Если в такой ситуации вам надо сообщить пользователю о необходимости установить DirectX нужной версии, то действуйте именно так, как мы рассмотрели выше. Описываемый далее способ, предлагаемый разработчиками, для такой ситуации не совсем хорош, поскольку опирается на принципиально новые функции, отсутствующие в библиотеках ранних версий DirectX. При попытке загрузки отсутствующей функции будет генерироваться исключение. Поэтому ваше приложение может просто не добраться до информирования пользователя.

Для старших версий DirectX разработчики рекомендуют пользоваться функцией

```
DirectDrawCreateEx : function (lpGUID: PGUID;
                               out lplpDD: IDirectDraw7; const iid: TGUID;
                               pUnkOuter: IUnknown) : HResult; stdcall;
```

Главный объект теперь должен быть типа `IDirectDraw7`, здесь же мы указываем требуемый нами интерфейс. То есть эта функция объединяет два действия, рассмотренные в предыдущем примере.

Очередным примером является проект каталога Ex05. Код немного отягощен включением защищенного режима, но приложение будет корректно работать на компьютере со старой версией DirectX.

Главный объект здесь имеет тип `IDirectDraw7`, а обработчик события `OnCreate` формы выглядит так:

```
procedure TForm1.FormCreate(Sender: TObject);
var
  hRet : HRESULT;   // Вспомогательная переменная для анализа результата
begin
  FDD := nil;       // Это обязательно для повышения надежности работы
  try               // Включаем защищенный режим
  try               // ...finally
  // Создание главного объекта DirectDraw
  hRet := DirectDrawCreateEx (nil, FDD, IDirectDraw7, nil);
  if Failed (hRet)  // В случае ошибки наверняка сюда не доберемся
     then ShowMessage ('DirectX 7-й версии не доступен')
     else ShowMessage ('DirectX 7-й версии доступен');
  finally           // В любом случае производим освобождение памяти
  if Assigned (FDD) then FDD := nil;
  end;
  except            // В случае ошибки информируем о неудаче
     ShowMessage ('DirectX 7-й версии не доступен')
  end;
end;
```

Как видно из комментариев, анализ значения переменной `hRet` здесь можно и не производить, обращение к функции `DirectDrawCreateEx` на компьютере с установленным DirectX версии младше седьмой приведет к появлению исключения.

В наших последующих примерах мы, как правило, будем пользоваться именно функцией `DirectDrawCreateEx`, чтобы иметь доступ ко всем возможностям, предоставляемым последними версиями DirectX. Так рекомендуют разработчики. Защищенный режим в такой ситуации включать не будем, но только в погоне за удобочитаемостью кода.

Что вы узнали в этой главе

Первая, вводная глава посвятила читателей в программную архитектуру операционной системы и напомнила о важной роли динамических библиотек в этой архитектуре. COM-модель будем считать развитием технологии "традиционных" DLL, позволяющей использовать парадигму ООП на уровне операционной системы, функций API.