СЕРГЕЙ ЕСЕНИН

bhv®

DirectX и Delphi:

РАЗРАБОТКА ГРАФИЧЕСКИХ И МУЛЬТИМЕДИЙНЫХ ПРИЛОЖЕНИЙ

КОМПОНЕНТЫ DirectX: DirectX Graphics, DirectSound, DirectMusic, DirectInput, DirectShow

ШЕЙДЕРЫ И ЯЗЫК HLSL

ДВУМЕРНАЯ И ТРЕХМЕРНАЯ ГРАФИКА

ЗАХВАТ И ВОСПРОИЗВЕДЕНИЕ ИЗОБРАЖЕНИЯ И ЗВУКА

РАБОТА С УСТРОЙСТВАМИ ВВОДА

PRO

ПРОФЕССИОНАЛЬНОЕ ПРОГРАММИРОВАНИЕ

+CD

Сергей Есенин

DirectX и Delphi:

РАЗРАБОТКА ГРАФИЧЕСКИХ И МУЛЬТИМЕДИЙНЫХ ПРИЛОЖЕНИЙ

Санкт-Петербург
«БХВ-Петербург»
2006

УДК 681.3.06
ББК 32.973.26-018.2
Е82

Есенин С. А.

Е82 DirectX и Delphi: разработка графических и мультимедийных приложений. — СПб.: БХВ-Петербург, 2006. — 512 с.: ил.

ISBN 5-94157-867-9

Рассмотрена разработка приложений с использованием технологии DirectX в среде программирования Borland Delphi. Подробно описаны все основные компоненты, входящие в состав DirectX: DirectX Graphics, DirectShow, DirectInput, DirectSound и DirectMusic. Показано создание собственных наборов классов, облегчающих работу с различными компонентами DirectX. На практических примерах рассмотрена работа с двумерной и трехмерной графикой, шейдеры и язык HLSL, различные цветовые эффекты, работа с текстурой, освещением и т. д. Уделено внимание выводу изображения в оверлейном режиме, механизмам захвата изображения (на примере работы с web-камерой) и захвата звука. Представлены механизмы воспроизведения мультимедиаданных в различных форматах: AVI, MPEG, MP3 и др. Прилагаемый компакт-диск содержит исходные коды примеров, рассмотренных в книге, а также набор классов.

Для программистов

УДК 681.3.06
ББК 32.973.26-018.2

Группа подготовки издания:

Главный редактор	*Екатерина Кондукова*
Зам. главного редактора	*Игорь Шишигин*
Зав. редакцией	*Григорий Добин*
Редактор	*Анна Кузьмина*
Компьютерная верстка	*Натальи Смирновой*
Корректор	*Наталия Першакова*
Дизайн серии	*Инны Тачиной*
Оформление обложки	*Елены Беляевой*
Зав. производством	*Николай Тверских*

Лицензия ИД № 02429 от 24.07.00. Подписано в печать 31.05.06.
Формат $70\times100^{1}/_{16}$. Печать офсетная. Усл. печ. л. 41,28.
Тираж 2500 экз. Заказ №
"БХВ-Петербург", 194354, Санкт-Петербург, ул. Есенина, 5Б.

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

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

ISBN 5-94157-867-9

© Есенин С. А., 2006
© Оформление, издательство "БХВ-Петербург", 2006

Оглавление

Благодарности

Данная книга вышла в свет благодаря моральной поддержке и помощи со стороны моей жены Ольги и родителей, к которым я и приехал от суеты городской в деревню, дабы в спокойной обстановке закончить то, что начал.

Также благодарю за помощь своего товарища по работе, Хворова Василия. Его советы и замечания оказались очень ценными.

Конечно же, стоит отметить и роль сотрудника издательства "БХВ-Петербург" Шишигина Игоря, всеми правдами и неправдами заставившего меня проделать эту работу. За что ему отдельное спасибо!

Введение

Работая за компьютером с установленной операционной системой Microsoft Windows, мы, не задумываясь, можем запустить какую-либо трехмерную игру, в которой будем управлять, к примеру, самолетом при помощи джойстика или мыши и клавиатуры. Или решим послушать музыку, или просмотреть какой-то интересный фильм. Можем пообщаться по сети в режиме реального времени, да еще при этом получая изображения собеседника с Web-камеры или иного устройства захвата. И все это и даже больше нам помогает проделать система DirectX.

Свою книгу я решил посвятить описанию приемов разработки графических и мультимедиаприложений с использованием системы DirectX применительно к среде разработке Borland Delphi. Почему именно Delphi, а не Borland C++ Builder или, скажем, Microsoft Visual Studio? Да хотя бы потому, что книг по программированию с использованием DirectX в Delphi не так много. А отдельные компоненты DirectX, такие как DirectShow, вообще мало освещены.

В книге я постарался раскрыть такие аспекты разработки программного обеспечения с использованием DirectX, как работа с двумерной и трехмерной графикой, работа со звуком, устройствами ввода и мультимедиапотоками. Описание работы с графикой (подсистема DirectX Graphics) состоит из двух частей — описание подсистемы Direct3D и подсистемы DirectDraw. Несмотря на то, что подсистема DirectDraw считается несколько устаревшей, она все равно не утратила своей актуальности, и ее интерфейсы будут поддерживаться в DirectX и в дальнейшем. Работа со звуком будет изучена в главах, описывающих работу с DirectSound и DirectMusic; работа с устройствами ввода, такими как клавиатура, мышь и джойстик, будет рассмотрена в главе, посвященной DirectInput. А в последней части книги мы проанализируем работу с мультимедиапотоками: научимся воспроизводить такие мультимедиаформаты, как AVI, MPEG, MP3 и т. д. Научимся получать изображение и звук с устройств захвата и сохранять на диске.

На кого рассчитана книга

Книга в первую очередь рассчитана на людей, знакомых со средой разработки Delphi и имеющих представление о технологии COM, которые хотят изучить систему DirectX и такие ее возможности, как работа с графикой, устройствами ввода, работа с мультимедиаданными и т. д.

Предполагается, что читателю не нужно объяснять всех тонкостей работы в среде Delphi и всех тонкостей технологии COM. Тем не менее, в первой части книги среда разработки и возможности COM будут кратко описаны, и даже будет приведен пример разработки COM-сервера и клиентской части.

Как построена книга

Книга состоит из данного введения, шести частей, заключения, двух приложений, списка литературы и предметного указателя. В свою очередь шесть частей содержат пятнадцать глав.

Часть I книги является вводной. В ней представлены основные сведения о системе DirectX, расписаны компоненты, входящие в ее состав. Дается сравнение DirectX с OpenGL. Кратко описывается среда разработки Borland Delphi. Рассматриваются возможности библиотеки COM. Дается описание интерфейсов и COM-объектов, уникальных идентификаторов и результата вызова методов. В заключение приводится пример: пишем COM-сервер и клиентскую часть.

Часть II содержит описание графической подсистемы DirectX Graphics. Первая половина предлагает описание работы подсистемы Direct3D. Мы обсудим работу в оконном и полноэкранном режимах. Научимся рисовать различные примитивы, строить различные фигуры из примитивов на плоскости и в пространстве. Подробно изучим работу с матрицами и разберем, какие типы матриц используются в Direct3D, и для чего каждая из них предназначена. На практике научимся строить трехмерные объекты из примитивов на примере куба, изучим свойства освещения и материалов, научимся использовать туман и работать с текстурой. Изучим различные типы фильтрации текстур, такие как линейная фильтрация, анизотропная и многоуровневая фильтрации. Выясним возможности библиотеки утилит D3DX и рассмотрим mesh-объекты. Научимся работать с текстом на плоскости и в пространстве, накладывать на объект несколько текстур, а также изучим мультитекстурирование на примере. Обсудим возможность создания прозрачных объектов различными способами и научимся использовать эффект размытия при движении (Motion Blur). Разберем, что такое шейдеры, и научимся писать их на языке HLSL.

Вторая половина этой части вкратце описывает возможности и приемы работы с подсистемой DirectDraw. Будет проведен обзор библиотеки, ее возможностей, достоинств и недостатков. Мы изучим порядок работы с данной подсистемой, научимся работать с ней в полноэкранном и оконном режимах, рассмотрим различные типы поверхностей DirectDraw. Узнаем, что такое цветовой ключ и зачем он нужен, обсудим работу с палитрой, научимся

работать с поверхностью напрямую и разберемся, как работать с оверлейными поверхностями.

Основным отличием этой части от последующих является то, что изучение подсистем Direct3D и DirectDraw построено по принципу примеров, т. к. это наиболее простой и удобный способ изучения работы с графикой. В дальнейшем, в каждой части книги для описываемой подсистемы будут созданы собственные классы и примеры их использования.

Часть III описывает работу с подсистемой DirectSound. Мы рассмотрим область применения данной подсистемы, ее достоинства и недостатки, изучим принцип работы. Разберем, что такое уровни взаимодействия и потеря буферов. Научимся воспроизводить WAV-файлы и накладывать на звук различные эффекты. Будет представлен класс, упрощающий работу с DirectSound, под названием `TdxSoundManager`, и рассмотрен пример его использования. Затем мы изучим способы захвата звука и записи в WAV-файл. Для этого нами будет рассмотрен класс `TdxSoundCapture` вместе с примером.

Часть IV расскажет нам о подсистеме DirectMusic и ее отличиях от DirectSound. С помощью нее мы научимся воспроизводить MIDI- и WAV-файлы. Будут представлены классы `TdxMusicSegment` и `TdxMusicManager`, описывающие звуковой сегмент и менеджер воспроизведения соответственно.

Часть V описывает работу с устройствами ввода. Мы изучим режимы работы и уровни взаимодействия и разберем общий алгоритм работы. Рассмотрим класс `TdxInputManager`, упрощающий работу с клавиатурой, мышью и джойстиком, и пример его использования.

Часть VI содержит информацию о подсистеме DirectShow. Это архитектура, позволяющая управлять потоками мультимедиаданных. Сначала мы рассмотрим область применения данной архитектуры и поддерживаемые форматы потоков данных. Изучим такие понятия, как фильтр, граф фильтров и менеджер графа фильтров. Рассмотрим способы воспроизведения таких мультимедиаформатов, как AVI, MPEG, MP3 и др. Разберем работу интерфейсов управления фильтром, контактом, работу интерфейса графа фильтров и интерфейса управления графом фильтров. Рассмотрим интерфейсы управления позиционированием в потоке, управления выводом звука, управления механизмом событий, управления выводом видеоданных и интерфейс перехвата кадра из потока видео. Для работы с подсистемой DirectShow будет представлен класс `TdxMediaPlayer`, который фактически инкапсулирует набор свойств и методов, характерных для мультимедиапроигрывателя, а пример использования класса и будет тем самым проигрывателем.

Затем перейдем к изучению архитектуры захвата изображения и звука и разберем работу всех нужных нам интерфейсов. Научимся перечислять устройства определенных классов. Рассмотрим режимы предварительного просмотра и захвата потоков данных. Изучим возможность захвата изображения и звука одновременно, а также научимся сжимать полученные данные. Обсудим возможность настройки устройств с помощью страниц свойств и рассмотрим общую последовательность шагов, необходимых для получения данных с различных устройств захвата. Класс `TdxCaptureManager`, который будет рассмотрен нами в конце части, обеспечивает возможность захвата и предварительного просмотра видеопотока и потока аудио одновременно.

В *заключении* будет подведен краткий итог книги, а также представлена информация, как можно связаться с автором книги.

Приложения, которые представлены в конце книги, содержат информацию о наиболее интересных интернет-ресурсах, которыми, так или иначе, пользовался в свое время автор книги и пользуется сейчас, и описание прилагаемого к книге компакт-диска.

В книге также присутствует список литературы, который поможет читателю найти дополнительную информацию.

Ну и последнее — это предметный указатель. Он представляет собой наиболее удобный инструмент для поиска в книге по ключевым словам.

Требования к компьютеру и программному обеспечению

Для обеспечения корректной работы всех примеров, приведенных в книге, рекомендуется следующая конфигурация компьютера:

- процессор Intel Pentium III 1000 МГц и выше;
- видеокарта 32 Мбайт (1024×768) и более производительная, поддерживающая работу с DirectX 8.0 и выше;
- оперативная память 128 Мбайт и выше;
- жесткий диск объемом 10 Гбайт и более;
- CD/DVD-привод;
- операционная система Microsoft Windows 2000/XP/Server 2003;
- DirectX 9.0;
- установленная среда разработки Borland Delphi 7 и старше.

Не следует считать приведенную конфигурацию компьютера окончательной. Примеры будут работать и на компьютере с меньшей производительностью.

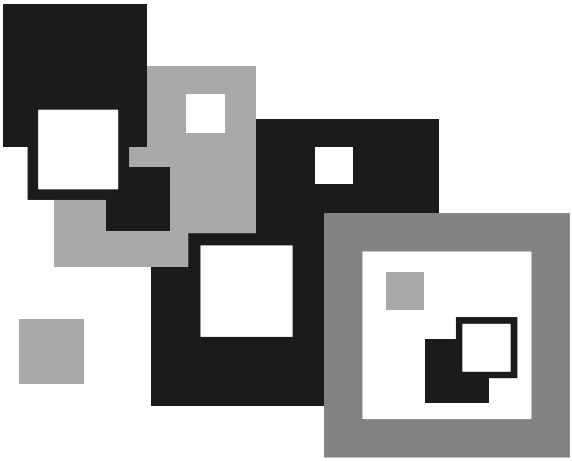
Однако требования к DirectX на компьютере не меняются — должна быть установлена версия не ниже версии DirectX 9.0.

На прилагаемом к книге компакт-диске *(см. приложение 2)* располагаются необходимые для сборки примеров заголовочные файлы DirectX, а также библиотеки, необходимые для работы с графической подсистемой Direct3D.

Все примеры были протестированы, как минимум, на трех компьютерах следующей конфигурации:

- Intel Pentium IV 2,4 ГГц\512 Мбайт DDR\128 NVidia FX 5200\120 Гбайт HDD Maxtor\DVD-RW Nec\Windows XP SP2\DirectX 9.0c;
- Intel Pentium IV 3,0 ГГц\1024 Мбайт DDR\128 ATI Radeon 9250\80 Гбайт HDD Seagate\DVD-Rom Toshiba\Windows XP SP2\DirectX 9.0c;
- ноутбук BLISS 507S: Intel Pentium M 1,73 ГГц\512 Мбайт DDR\128 ATI Radeon X700\60 Гбайт HDD\DVD-RW\Windows XP SP2\DirectX 9.0c.

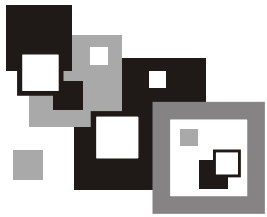
Для корректной работы примеров вы должны указать в настройках среды Borland Delphi путь к папке с заголовками DirectX, которые находятся в каталоге DirectX прилагаемого к книге компакт-диска. А в папке Lib хранятся библиотеки, необходимые для работы примеров с подсистемой Direct3D. Вам необходимо переписать библиотеки в такой каталог у себя на компьютере, к которому прописан путь в настройках Windows, например, Windows\System32.



ЧАСТЬ I

Общие сведения

Глава 1



DirectX

Система DirectX представляет собой большой набор API-функций низкого уровня, позволяющих разрабатывать различные высокопроизводительные графические и мультимедиаприложения. Имеются средства для работы со звуком, устройствами ввода и упрощена разработка сетевых приложений. Используется система зачастую в таких областях, как разработка компьютерных игр и систем безопасности.

Практически весь набор API-функций, так или иначе, базируется на технологии COM.

Состав DirectX

Свое знакомство с мощной системой, именуемой DirectX, мы начнем с изучения ее состава. Итак, вот основные компоненты (подсистемы), входящие в состав DirectX:

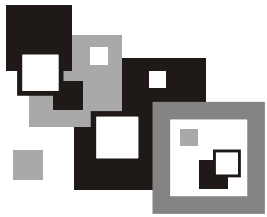
- DirectX Graphics — компонент, объединивший в себе две мощных графических подсистемы для работы с двумерной и трехмерной графикой — DirectDraw и Direct3D;
- DirectShow — архитектура, позволяющая управлять захватом и воспроизведением мультимедиапотоков;
- DirectInput — подсистема, используемая для работы с различными устройствами ввода, такими как клавиатура, мышь, джойстик, и другими игровыми устройствами (например, устройствами с обратной связью);
- DirectSound — компонент DirectX, обеспечивающий работу с оцифрованным звуковым потоком;

- DirectMusic — компонент DirectX, так же как и DirectSound обеспечивающий работу со звуковым потоком, только поддерживающий работу и с форматом MIDI;
- DirectPlay — подсистема, позволяющая разрабатывать многопользовательские приложения, ярким примером которых служат многопользовательские игры;
- DirectSetup — простой набор API-функций, позволяющий устанавливать компоненты DirectX одним вызовом;
- DirectX Media Objects (DMO) — базирующиеся на технологии COM компоненты поддержки потоковых объектов.

DirectX или OpenGL?

Как известно, помимо технологии Direct3D, для вывода трехмерной графики существует еще целый ряд технологий, одной из которых является OpenGL. Данная технология интересна своей поддержкой отличных от системы Microsoft Windows операционных систем, в то время как DirectX совместима только с ОС Windows. Собственно была даже своеобразная война между сторонниками этих систем.

Но на этом их конкурентоспособность и заканчивается: в системе OpenGL отсутствует поддержка работы со звуком, сетью, устройствами ввода и т. д. Зато отдельные компоненты DirectX прекрасно уживаются с OpenGL, что и восполняет все пробелы.



Глава 2

Среда разработки Borland Delphi

Изучать систему DirectX мы будем применительно к языку программирования Pascal, и работать будем в довольно популярной среде разработки Borland Delphi 7. Собственно версия особого значения не имеет, будь то более ранняя или поздняя версия — достаточно установить соответствующие версии заголовочных файлов DirectX и запустить наши примеры. При невозможности открытия проекта (несоответствие формата формы и т. п.) достаточно создать новый проект и перенести код в него.

Структура среды разработки

Среда разработки Borland Delphi 7 состоит из нескольких отдельно расположенных функциональных окон. К основным можно отнести следующие окна:

- палитра компонентов (**Component Palette**);
- дизайнер форм (**Form Designer**);
- дерево объектов (**Object TreeView**);
- инспектор объектов (**Object Inspector**);
- окно редактора кода (**Editor Window**).

Имеются также и различные окна отладки, настроек и т. д. Общий вид среды разработки Delphi после запуска представлен на рис. 2.1.

Сверху расположена палитра компонентов, под ней слева размещаются друг под другом дерево объектов и инспектор объектов. Справа от них находятся окно редактора кода и дизайнер форм.

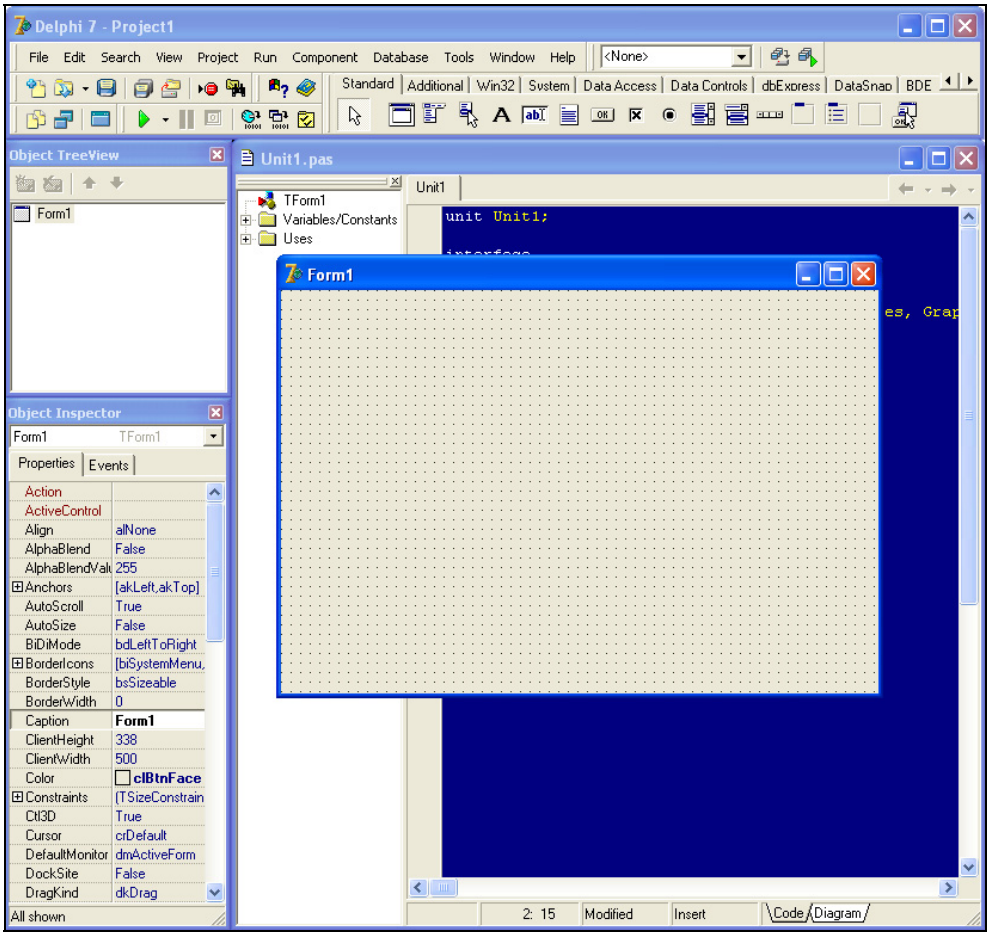


Рис. 2.1. Среда разработки Borland Delphi

Первое приложение

Создавать приложения в Delphi очень удобно и даже просто. Выберем пункт меню **File | New | Application**, и у нас появляется каркас приложения с готовым файлом проекта и одной формой (рис. 2.2). Сохранив проект на диск и нажав клавишу <F9>, мы откомпилируем и запустим наше приложение, состоящее из одной пустой формы. Как бы там ни было, это законченное приложение, пусть и абсолютно бесполезное с точки зрения функциональности, зато в дальнейшем такое простое создание окон сильно облегчит нашу жизнь.

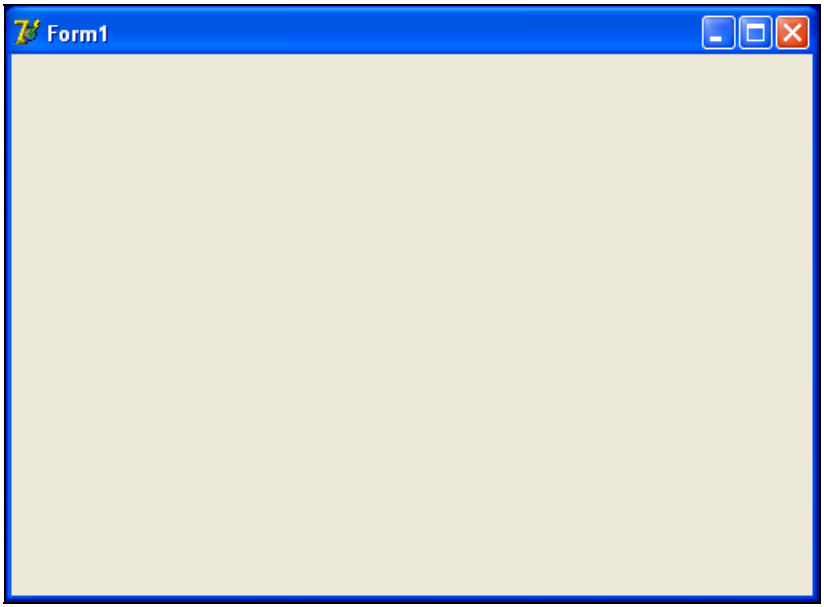
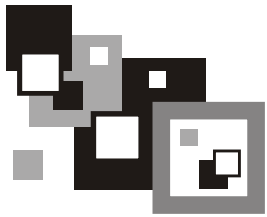


Рис. 2.2. Первое приложение

Однако не стоит думать, что работа с подсистемой DirectX настолько же проста, насколько и создание простейшего приложения в Delphi.

Глава 3



Библиотека COM

Так что же такое COM? Что это за библиотека и зачем она нужна? Почему DirectX базируется на COM?

Расшифровывается аббревиатура COM как Component Object Model — модель компонентных объектов. Модель не зависит от платформы, является распределенной объектно-ориентированной системой для создания бинарных интерактивных компонентов. И в связи с тем, что, как уже было сказано, DirectX базируется на COM, мы должны иметь хотя бы минимальное представление о данной библиотеке и уметь пользоваться ее возможностями. Несмотря на то, что библиотека COM покажется кому-то сложной и запутанной, в большинстве случаев ее использование в разработке приложений с помощью DirectX оказывается достаточно простым.

COM или DLL?

Библиотеки динамической компоновки (DLL, Dynamic Link Library) предоставляют схожую функциональность. Приложение подгружает библиотеку и может использовать ее функциональность в своих целях. Подобно DLL, объекты COM предоставляют методы, которые приложение может использовать по своему усмотрению. Взаимодействие с объектами COM происходит практически так же, как и работа с объектами Delphi. Но имеется и целый ряд отличий:

- объекты COM имеют более строгую инкапсуляцию. Мы не можем просто так создать COM-объект и использовать все его методы, т. к. все методы, так или иначе, сгруппированы по интерфейсам. Для вызова определенного метода нам может понадобиться создать COM-объект и запросить нужный интерфейс;

- объекты COM — это не объекты Delphi, и процесс их создания будет иным. Существует несколько способов создания COM-объектов, но все они используют методы библиотеки COM. API системы DirectX содержит ряд методов, которые упрощают создание некоторых объектов DirectX;
- для управления жизненным циклом объекта мы должны так же пользоваться методами библиотеки COM;
- COM-объекты не требуют явной загрузки. Обычно эти объекты COM так же располагаются в DLL, но нам не требуется загружать эту библиотеку или подключать ее статически для использования COM-объектов. Каждый COM-объект имеет свой уникальный идентификатор, который и используется для его создания. COM автоматически загружает нужную библиотеку DLL;
- поскольку модель COM представляет собой стандарт бинарной разработки для программных компонентов, то это означает независимость от языка разработки. Создаваемые объекты могут выполняться в одном процессе, в разных процессах и даже на другом компьютере.

Объекты и интерфейсы

Выше уже было упомянуто такое понятие, как *интерфейс*. Под интерфейсом понимается набор сгруппированных по определенным признакам методов. COM-объект — это реализация интерфейса (одного или нескольких одновременно). То есть фактически при вызове какого-либо метода интерфейса мы вызываем метод объекта. Отдельно следует упомянуть, что как один COM-объект может реализовывать произвольное количество интерфейсов, так и один интерфейс может реализовываться различными объектами COM.

В Delphi есть такое понятие, как *абстрактный метод*. Из таких методов строятся *абстрактные классы*. Интерфейсы и абстрактные классы очень схожи по своей сути, но имеют и ряд существенных отличий. Например, класс, являющийся производным, может реализовывать несколько интерфейсов, в то время как у него может быть только один базовый класс.

Интерфейс *IUnknown*

Так же как и класс Delphi имеет базовый класс `TObject`, так и для интерфейсов определен базовый интерфейс — `IUnknown`. По правде говоря, в Delphi, начиная с 6 версии, этот интерфейс именуется `IInterface`, что в принципе не меняет его сути. Для интерфейсов, в отличие от объектов, наследование не может означать повторного использования кода, т. к. интерфейс и его реализация — две абсолютно разные вещи. Также нужно ска-

зать, что наследование интерфейсов не может быть выборочным, т. е. производный интерфейс наследует все методы базового интерфейса.

Интерфейс `IUnknown` содержит всего три виртуальных метода. Первый — это получение указателя на интерфейс COM-объекта:

```
function QueryInterface(
  const IID: TGUID;
  out Obj):
HResult; stdcall;
```

Здесь:

- `IID` — уникальный идентификатор запрашиваемого интерфейса;
- `Obj` — переменная, в которую будет занесен запрашиваемый интерфейс. Если объект не поддерживает запрашиваемый интерфейс, то в переменную будет записано нулевое (`NIL`) значение.

Оставшиеся два метода управляют подсчетом ссылок. Увеличение числа ссылок на единицу:

```
function _AddRef: Integer; stdcall;
```

И уменьшение числа ссылок на объект на единицу:

```
function _Release: Integer; stdcall;
```

Методы управления подсчетом ссылок не требуется вызывать в явном виде — Delphi сделает это автоматически. Это означает, что при создании COM-объекта будет автоматически вызван метод `_AddRef`, а при присваивании указателю на интерфейс значения `NIL` (или когда объект выйдет за область видимости) автоматически будет вызван метод `_Release`.

Уникальные идентификаторы

Как вы уже наверно успели заметить, в методе `QueryInterface` первым параметром мы передаем некий уникальный идентификатор запрашиваемого интерфейса, имеющий тип данных `TGUID`.

Глобальные идентификаторы являются ключевой составляющей библиотеки COM. Если просто посмотреть на этот идентификатор, то это обычная структура (запись), состоящая из 128 битов. При создании идентификатора гарантируется его уникальность. COM широко использует эти идентификаторы для следующих целей:

- для уникальной идентификации COM-объекта. Такие идентификаторы называются *идентификаторами класса* (Class Identifier, CLSID). Они будут использоваться нами для создания конкретного COM-объекта;

- для идентификации определенного интерфейса. Значение GUID, которое определяет некоторый интерфейс, будет называться *идентификатором интерфейса* (Interface Identifier, IID).

Несмотря на то, что уникальный идентификатор представляет собой структуру, его нередко записывают в виде строки. Ее формат — это пять шестнадцатеричных целых чисел в формате `8-4-4-4-12` или `"{xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx}"`. Например, идентификатор интерфейса `IDirect3D9`:

```
{81BDCBCA-64D4-426d-AE8D-AD0147F4275C}
```

Идентификатор запомнить достаточно проблематично, и при его написании легко допустить ошибку. Гораздо проще использовать его эквивалентное имя. К примеру, это имя мы можем использовать для создания COM-объекта. По принятым соглашениям мы должны добавлять префиксы `IID_` или `CLSID_` к имени интерфейса или объекта. Например, для интерфейса `IDirect3D9` идентификатором будет выступать `IID_IDirect3D9`.

Результат *HResult*

Методы COM-объектов возвращают 32-битное целочисленное значение типа `HResult`. В большинстве случаев тип `HResult` представляет собой структуру, содержащую две основные информативные части:

- корректно ли отработал метод или произошла ошибка;
- более детальная информация о результате операции.

Можно использовать в качестве значения константы, описанные в модуле Windows.pas, такие как `S_OK`, `E_FAIL`, `E_UNEXPECTED`, `E_NOTIMPL` и т. д. Но можно использовать и собственные значения. Результаты вызовов методов COM-объекта обычно описываются в документации к ним.

Существует соглашение, по которому коды успешного завершения метода начинаются с префикса `S_`, а коды завершения с ошибкой — с префикса `E_`, например, `S_OK` и `E_FAIL`.

То, что методы могут возвращать различные варианты успеха или неудачи, означает, что нам нужно быть внимательными при анализе результата. К примеру, метод возвращает `S_OK` при успешном завершении работы и `E_FAIL` при ошибке. Тогда код обработки результата может выглядеть следующим образом:

```
if Result = E_FAIL then
  begin
    // Произошла ошибка, обрабатываем
  end
```

```
else begin
    // Ошибок нет
end;
```

А теперь допустим, что код ошибки может быть равен E_FAIL, E_UNEXPECTED, E_NOTIMPL и т. п. А у нас анализируется только E_FAIL, и все остальные ошибочные результаты будут обработаны так же, как и успешные. Это означает, что нам понадобится более детальный анализ всех возможных результатов.

Для облегчения нашей печальной участи в модуле Windows.pas определены 2 метода, которые тестируют результат HResult на предмет успеха или ошибки. Первый метод — Succeeded. Он проверяет, является ли результат успешным или нет:

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

Здесь Status — тестируемое значение.

Результатом вызова этого метода будет TRUE, если тестируемое значение является успешным результатом выполнения метода, и FALSE — в противном случае.

Второй метод — Failed, который полностью противоположен первому — он проверяет, является ли результат ошибочным или нет:

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

Результатом вызова этого метода будет TRUE, если тестируемое значение является ошибочным, и FALSE — в противном случае.

В своей работе мы достаточно часто будем применять эти два метода. Скажем, тот пример, который мы приводили ранее, должен быть исправлен следующим образом:

```
if FAILED(Result) then
begin
  // Произошла ошибка, обрабатываем
end
else begin
  // Ошибок нет
end;
```

Но не следует думать, что все методы COM-объектов возвращают тип HResult. Так, например, методы IUnknown._AddRef и IUnknown._Release возвращают текущее количество ссылок на объект.

Инициализация библиотеки COM и завершение работы с ней

Начинать работу с библиотекой COM необходимо с инициализации. Для этого имеется метод CoInitializeEx, описанный в модуле ActiveX.pas:

```
function CoInitializeEx(
  pvReserved: Pointer;
  coInit: Longint):
HResult; stdcall;
```

Здесь:

- pvReserved — зарезервировано. Должно использоваться нулевое значение;
- coInit — флаг, определяющий потоковую модель:
  - COINIT_MULTITHREADED — многопоточная модель — объекты могут вызываться из разных потоков;
  - COINIT_APARTMENTTHREADED — раздельное адресное пространство у потоков;
  - COINIT_DISABLE_OLE1DDE — отключение поддержки DDE для OLE1;
  - COINIT_SPEED_OVER_MEMORY — использование большего объема памяти для увеличения быстродействия.

Этот метод производит инициализацию статических и загружаемых библиотек COM и устанавливает текущую потоковую модель. Он должен быть вызван перед началом использования функций COM API (кроме функции CoGetMalloc и функций распределения памяти).

Для завершения работы с COM необходимо вызвать метод CoUninitialize, который освобождает ресурсы загруженных библиотек:

```
procedure CoUninitialize; stdcall;
```

Вызов метода будет иметь успех только тогда, когда перед ним был произведен вызов метода инициализации библиотеки COM CoInitializeEx. И наоборот, если был произведен вызов CoInitializeEx, то вызов CoUninitialize обязателен.

Глава 4

Пишем COM-сервер

Создание COM-сервера

В этой главе мы с вами создадим первый COM-сервер, научимся его регистрировать и использовать все его возможности в своем приложении. В роли COM-сервера будет выступать так называемый In-Process COM Server, реализованный в виде DLL.

Для создания такой библиотеки в Delphi необходимо сформировать библиотеку ActiveX. Выберем пункт меню **File | New | Other...**, и у нас на экране появится диалог выбора создаваемого объекта. Перейдем на вкладку **ActiveX** и выберем пункт **ActiveX Library** (рис. 4.1).

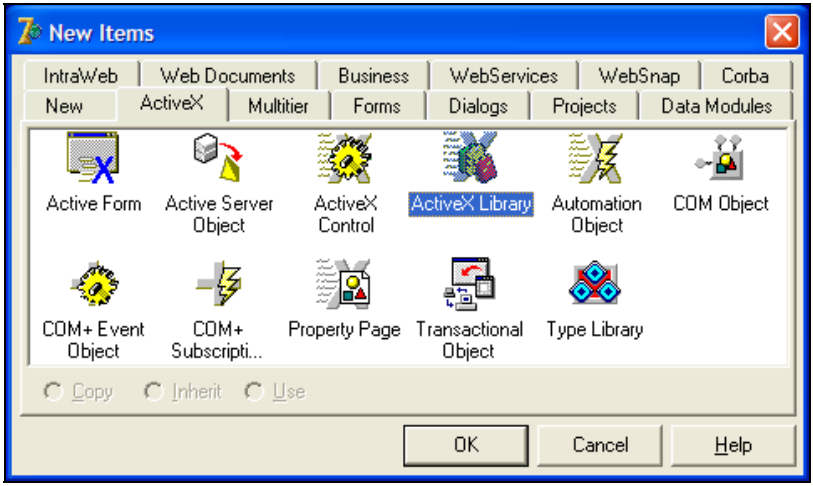


Рис. 4.1. Диалог создания нового объекта

Сохраним проект под именем COM_Server. Весь текст нашего модуля состоит всего из нескольких строк (листинг 4.1).

Листинг 4.1. Текст модуля COM_Server.dpr

```
library COM_Server;

uses
  ComServ;

exports
  DllGetClassObject,
  DllCanUnloadNow,
  DllRegisterServer,
  DllUnregisterServer;

{$R *.RES}

begin
end.
```

Как мы видим, в модуле присутствуют четыре экспортируемые функции:

- `DllGetClassObject` — получение класса объекта;
- `DllCanUnloadNow` — проверка возможности выгрузки COM-сервера из памяти;
- `DllRegisterServer` — регистрация COM-сервера в системном реестре;
- `DllUnregisterServer` — удаление из реестра информации о COM-сервере.

Для нас нет необходимости в реализации данных функций, т. к. они уже и так реализованы в модуле ComServ.pas.

Теперь необходимо собственно создать код COM-сервера. Для примера я хотел бы реализовать следующую идею: мы создадим объект `TSimpleObject`, который будет реализовывать два интерфейса — интерфейс конфигурирования `ISimpleConfigurator` и интерфейс рисования `ISimpleDrawing`. Так же как и для создания ActiveX-библиотеки, нам нужно выбрать пункт меню **File | New | Other...** и перейти на вкладку **ActiveX**. Далее следует выбрать объект **COM Object** и нажать кнопку **OK**. У нас на экране появится мастер создания COM-объектов (COM Object Wizard). Зададим параметры COM-объекта,