ДЕНИС БУТОРИН

bhv

# MS Agent и Speech API в DELPHI

МЕТОДЫ, СВОЙСТВА И СОБЫТИЯ Microsoft Agent

ТЕХНОЛОГИИ СИНТЕЗА И РАСПОЗНАВАНИЯ РЕЧИ TEXT-TO-SPEECH И SPEECH RECOGNITION

РЕЧЕВЫЕ ИНТЕРФЕЙСЫ ВЫСОКОГО И НИЗКОГО УРОВНЕЙ

СОЗДАНИЕ СОБСТВЕННОГО АНИМИРОВАННОГО ПЕРСОНАЖА

PRO

+CD

Денис Буторин

# MS Agent и Speech API в DELPHI

Санкт-Петербург
«БХВ-Петербург»
2005

УДК 681.3.06
ББК 32.973.26-018.1Delphi
Б93

**Буторин Д. Н.**

Б93 MS Agent и Speech API в Delphi. — СПб.: БХВ-Петербург, 2005. — 448 с.: ил.

ISBN 5-94157-502-5

Рассмотрено программирование нестандартных пользовательских интерфейсов в среде Delphi с применением технологий Microsoft Agent и Microsoft Speech API для операционных систем Windows 98/2000/XP. Представлены способы внедрения анимированных персонажей в приложения Delphi и использования методов синтеза и распознавания речи с помощью функций Speech API. Подробно рассмотрены речевые интерфейсы высокого и низкого уровней. Описан процесс создания собственных анимированных персонажей и использования нестандартной текстовой выноски BalloonDialog.

На компакт-диске помещены все примеры, описанные в книге, необходимые компоненты и модули для создания приложений, а также программы автора, созданные с использованием описанных в книге технологий.

*Для программистов*

УДК 681.3.06
ББК 32.973.26-018.1Delphi

**Группа подготовки издания:**

| | |
|---|---|
| Главный редактор | *Екатерина Кондукова* |
| Зам. главного редактора | *Игорь Шишигин* |
| Зав. редакцией | *Григорий Добин* |
| Редактор | *Галина Смирнова* |
| Компьютерная верстка | *Натальи Смирновой* |
| Корректор | *Наталия Першакова* |
| Дизайн серии | *Инны Тачиной* |
| Оформление обложки | *Игоря Цырульникова* |
| Зав. производством | *Николай Тверских* |

Лицензия ИД № 02429 от 24.07.00. Подписано в печать 25.11.04.
Формат 70×100 1/16. Печать офсетная. Усл. печ. л. 36,12.
Тираж 3000 экз. Заказ №
"БХВ-Петербург", 190005, Санкт-Петербург, Измайловский пр., 29.

Гигиеническое заключение на продукцию, товар № 77.99.02.953.Д.001537.03.02
от 13.03.2002 г. выдано Департаментом ГСЭН Минздрава России.

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

ISBN 5-94157-502-5

# Содержание

# Введение

## О чем и для кого эта книга

Предметом книги является подробное изложение принципов программирования нестандартных пользовательских интерфейсов в среде Delphi с применением таких технологий, как Microsoft® Agent Character v2.0 (MS Agent) и Microsoft® Speech API 4.0 (SAPI 4.0) в операционных системах Windows 98, Windows 2000 и Windows XP. В ней рассказывается о способах внедрения анимированных персонажей MS Agent в приложения Delphi и о методах синтеза речи с помощью функций SpeechAPI. Книга содержит описание отличительных особенностей программирования в разных версиях Delphi. Кроме того, здесь описаны другие дополнительные компоненты и модули, используемые в технологии MS Agent. На прилагаемом компакт-диске помещены все описанные в книге примеры, а также компоненты и модули, необходимые для создания приложений с использованием технологий анимированных виртуальных персонажей MS Agent и синтеза речи средствами SpeechAPI.

Технология MS Agent создана корпорацией Microsoft и впервые была применена в приложениях MS Office. Наверняка все знают забавных персонажей MS Word — Скрепку, Мурку и т. п. Однако автор уверен, что не только Microsoft нравятся эти помощники. Некоторые программисты, особенно начинающие, хотели бы научиться использовать технологию анимированных персонажей в своих собственных приложениях, а MS Agent может пригодиться как в больших офисных приложениях в качестве интеллектуального помощника, так и в игровых программах, различных напоминалках и т. п.

Технология синтеза и распознавания речи Text-To-Speech и Speech Recognition также разработана компанией Microsoft®. В этой области имеются большие достижения, и, очевидно, развитие ее будет продолжаться. Не исключено, что сторонние разработчики пожелают применить подобные технологии в своих приложениях.

Существует немало англоязычных материалов по этой теме в применении к программированию на языках C/C++ и Visual Basic, а вот подробной документации для программистов Delphi недостаточно. Данная книга претендует на то, чтобы заполнить создавшийся пробел. Автор считает, что язык Delphi (начиная с версии 7 он называется так официально) ничем не хуже C/C++, и что, пользуясь им, можно осуществить почти все замыслы. Естественно, для каждого языка программирования определен конкретный круг решае-

мых задач, но это не значит, что для каких-либо из них допустимо отсутствие документации по узким темам.

Предлагаемая вашему вниманию книга представляет собой наиболее полное руководство по программированию в среде MS Agent, а также первую документацию на русском языке по использованию технологии SAPI 4.0.

Книга рассчитана на широкий круг читателей — от начинающих программистов Delphi до достаточно подготовленных. Тем не менее, автор надеется, что читатель уже знаком с объектно-ориентированным программированием (ООП) и базовыми понятиями Delphi.

## Структура и особенности книги

Книга состоит из четырех глав, построеных таким образом, чтобы их можно было читать независимо друг от друга. Тем не менее, в книге встречаются примеры программных кодов, которые связаны между собой (например, в *гл. 4*, в описании совмещения технологии MS Agent и SpeechAPI). Поэтому, если вы никогда раньше не работали с той или иной технологией, вам следует читать эти главы по порядку.

Глава 1 "Технологии COM, OLE и ActiveX" посвящена основам, необходимым для понимания технологий COM (Component Object Model) и ActiveX, а также началам программирования компонентов COM в Delphi. Она включает минимальный набор базовых знаний по COM-технологии и не заменяет собой какой-либо другой подробной и детальной документации. Если вы новичок в Delphi и (или) никогда не слышали о технологиях автоматизации, то вам необходимо прочитать эту главу, чтобы далее свободно понимать механизмы работы MS Agent и SpeechAPI. Если же вы представляете себе устройство COM или считаете, что владеете данной темой, то можете перейти к следующим главам.

В главе 2 "Технология MS Agent в среде Delphi" вам предстоит первое знакомство с персонажем MS Agent. Здесь рассказывается о его установке в операционной системе и среде программирования. Детально описываются практически все его интерфейсы, свойства, события и методы.

Глава 3 "Технология Speech API в среде Delphi" познакомит читателей с технологией SAPI4, с ее основными интерфейсами прикладного программирования: Voice Text API, Voice Command API, Voice Dictation API, Direct Text-To-Speech API и Direct Speech Recognition API. Здесь также рассказывается об основных компонентах, которые необходимы при использовании речевых технологий. В этой главе мы научимся писать приложения, синтезирующие речь, записывающие ее в аудиофайл. Мы рассмотрим в ней также вопросы распознавания речи с помощью SAPI и займемся созданием приложений, распознающих отдельные голосовые команды и поддерживающих диктовку.

Глава 4 "«Примочки» к MS Agent" представляет собой описание некоторых наиболее интересных дополнительных к MS Agent модулей и компонентов (так называемых "примочек"). Это готовые программные модули, приложения или ActiveX-компоненты, расширяющие функциональность персонажа. В этой главе мы научимся создавать свои собственные анимированные персонажи, подготавливать специальные лингвистические файлы для улучшения воспроизведения волновых файлов приложением MS Agent. Здесь же мы познакомимся с наиболее популярным ActiveX-компонентом для персонажа-агента BalloonDialog© от фирмы SommyTech®.

На прилагаемом компакт-диске записаны все примеры, обсуждаемые в книге с учетом различных версий Delphi. Те примеры, которые хорошо переносятся на Delphi 6 и Delphi 7, написаны в 5-ой версии среды. Программный код, который требует хотя бы незначительных поправок, дан в вариантах для других версий среды программирования. Кроме того, компакт-диск содержит все необходимые модули для написания приложений с использованием MS Agent и Speech API.

# Программные и аппаратные требования

Для работы с MS Agent рекомендуется иметь операционную систему Windows начиная с версии 95 и выше и приложение Internet Explorer версии v3.02 и выше. Необходимы: процессор модификации не ниже Pentium 100 МГц, объем оперативной памяти не менее 16 Мбайт, а также Windows-совместимая звуковая карта. Из дополнительных модулей для пользователей Windows 9*x* необходимы MS Agent Control v2.0 и один или несколько файлов персонажей. Ссылки вы найдете в *приложении 1.*

Для работы с синтезом и распознаванием речи *(см. гл. 3)* приведенные минимальные программные требования немного повышаются. Хотя это, в принципе, неважно, т. к. программное и аппаратное обеспечение компьютеров даже у рядовых пользователей на сегодняшний момент вполне удовлетворяет этим требованиям. Из дополнительных модулей необходимо установить набор функций Speech API v4.0 и один или несколько речевых синтезаторов (движков). Так как мы попытаемся распознавать речь, то ко всем установленным модулям надо добавить движки распознавания речи.

При изучении материала четвертой главы для создания собственных анимированных существ потребуется редактор персонажей Microsoft Agent Character Editor, а при рассмотрении вопросов создания лингвистических файлов — лингвистический редактор Microsoft Agent Linguistic Information Sound Editing Tool. Ну и, естественно, вам понадобится компонент BalloonDialog 6.5 для изучения примеров, иллюстрирующих работу "шаровоздушной" выноски.

# Глава 1

# Технологии COM, OLE и ActiveX

В данной главе будут рассмотрены основы COM-модели и ее спецификации. Мы познакомимся с объявлением интерфейсов в Delphi и научимся ими пользоваться. Затем напишем пример с размещением COM-объекта в DLL-библиотеке и в завершение познакомимся с технологией ActiveX и OLE Automation, создав простейший ActiveX-компонент и OLE Automation Server.

Начинающему программисту в Delphi необходимо прочитать эту главу для дальнейшего понимания всех конструкций и технологии решений в последующих главах. Однако данный материал не является заменой специальной литературы по COM-технологии. Он всего лишь позволит познакомиться с реализацией COM-технологии в теории и на практике.

## 1.1. Технология Component Object Model

С тех пор как появилось программирование, перед разработчиками больших проектов стоит задача обеспечения связи между отдельными программными модулями. Для решения этой нетривиальной задачи было разработано множество средств. Ко многим из них мы уже давно привыкли, например, к буферу обмена, разделяемым файлам, технологии динамического обмена данными (DDE — Dynamic Data Exchange). Технология OLE (*Linking and Embedding* — связывание и внедрение объектов) была разработана для обеспечения обмена данными и предоставления служб. Она решает общую задачу предоставления программными модулями друг другу собственных функций.

Существует множество других разработок, но во всех них в основе лежит базовая технология *Component Object Model* (COM) — многокомпонентная модель объектов или модель объектных компонентов. Это одна из главных технологий, на которых основывается Windows. С помощью COM один программный модуль предоставляет свои функции, а другие части программного обеспечения получают доступ к ним. Эти части могут находиться

как в одном процессе, так и в разных, и даже на различных компьютерах. Причем совершенно неважно, на каком из языков программирования реализованы эти части — при соблюдении спецификации COM они будут работоспособны в любой ситуации.

На основе COM разработаны такие технологии, как ActiveX, ActiveForm, Microsoft Transaction Server, DirectX, Shell и др.

COM является весьма успешной технологией, потому что позволяет строить мощные распределенные системы с минимумом усилий. Очевидные ее достоинства — в способности приложения COM развиваться с течением времени, удобстве и гибкости при модернизации существующих приложений.

Среда Delphi предоставляет разработчикам возможность создавать полноценные приложения, использующие технологию COM.

## 1.2. Базовые понятия

Рассмотрим базовые понятия, знание которых необходимо для работы с технологией COM.

### Объект

Ключевым понятием модели COM является *объект*. Это естественно, т. к. со времени появления объектно-ориентированного программирования (ООП) все сложные системы разбиваются на классы, экземпляры которых обладают собственными свойствами и методами. COM-компонент должен удовлетворять некоторым требованиям, приведенным ниже.

- Он должен скрывать используемый язык программирования.
- Объекты должны распространяться в двоичной форме, без необходимости компиляции и компоновки, т. е. готовыми к использованию.
- Должна быть предусмотрена возможность модернизировать компоненты, не затрагивая уже существующих пользователей. Новые версии компонента должны работать как со старыми, так и с новыми клиентами.
- Компоненты должны перемещаться по сети. Необходимо, чтобы компонент и использующий его клиент могли выполняться внутри одного процесса, в разных процессах и на разных машинах.

Итак, COM-приложение использует COM-объекты для предоставления своих функций клиентам. Стало быть, приложение должно иметь как минимум один COM-объект для того, чтобы считаться приложением COM.

Как и любой объект, COM-объект соответствует почти всем требованиям ООП. Он является экземпляром какого-либо класса, т. е. представляет собой переменную объектного типа. COM-объект имеет набор методов и

свойств и удовлетворяет трем основным требованиям: *инкапсуляции* (скрытие данных от прямого доступа), *полиморфизму* (свойство вызова методов в соответствии с классом фактически переданного параметра), а также некоторым элементам *наследования*.

## Интерфейс

Что делает COM-объект объектом COM, что заставляет его отличаться от остальных объектов? Как бы мы поступили, разрешая противоречие об обмене данными между приложениями на месте разработчиков COM?

Чтобы установить обмен данными между двумя разнородными системами (между клиентом и сервером), необходимо создать нечто общее, т. е. заранее "объяснить" компонентам, как они будут общаться. Это осуществляется с помощью одного из центральных элементов модели COM, называемого *интерфейсом*.

Интерфейс в модели COM — это средство, которое позволяет клиенту правильно обратиться к объекту COM, а объекту позволяет ответить клиенту так, чтобы он (клиент) его (сервер) понял. В идеологии COM каждый объект может поддерживать один или более интерфейсов, а каждый интерфейс — включать в себя несколько методов объекта для обеспечения доступа к его свойствам и выполнения определенных действий.

> ***Что такое интерфейс?***
>
> Вообще говоря, в самом широком смысле для интерфейсов любого типа существует неофициальное определение: интерфейс — это *то*, посредствам чего *что-то* соединяется с *чем-то*. Например, интерфейсом между нами и клавиатурой компьютера являются наши руки. Они позволяют нам — послать сообщение серверу (клавиатуре) о нажатии клавиши, а клавиатуре — вернуть клиенту (нам) ответ о том, что клавиша была нажата. Оригинальный пример, неправда ли? Я специально его придумал, чтобы читатель, незнакомый с COM, быстро запомнил это очень важное понятие: "интерфейс".

Строгое определение интерфейса (в применении к COM) в литературе может иметь следующую формулировку.

*Интерфейс* — это описание на языке программирования виртуальной таблицы адресов функций, поддерживаемых объектом. Подобные виртуальные таблицы аналогичны виртуальным таблицам методов (VMT — virtual method table) классов Delphi.

Каждый интерфейс имеет два *атрибута*. Первый атрибут — собственно название интерфейса, составленное в соответствии с правилами языка программирования. Имя должно начинаться с буквы "I" — так принято, так же как в Delphi название любого класса должно начинаться с буквы "T".

Второй атрибут представляет собой *глобальный уникальный идентификатор* (Globally Unique IDenitfier, GUID), который задается уникальным сочетанием символов. Для его генерации используются такие сильные алгоритмы, что однажды сгенерированное сочетание никогда не повторится в будущем ни на одном компьютере мира.

Существует особенность в характере наследования у объектов COM. В случае наследования интерфейса код реализации должен представить потомок, а наследование реализации подразумевает полное включение программного кода родителя в код потомка. Однако реализация при первом варианте наследования никуда не исчезает, объекты используют механизм включения и при необходимости потомок вызывает метод родителя.

Если интерфейс представляет собой своеобразную палочку-выручалочку для получения доступа к методам объекта, то как же тогда к ним обратиться? Для этого необходимо получить указатель на соответствующий интерфейс, после чего клиент имеет право использовать службы объекта, вызывая его методы как методы обычного объекта.

Следует еще пояснить некоторые детали. Поскольку объект может имеет несколько интерфейсов, то при получении интерфейса для каждого из них будет получен собственный указатель. Иначе говоря, при вызове процедур получения интерфейса вы получаете не один указатель для всех интерфейсов объекта, а только тот, который был запрошен. Это удобно, потому что гораздо рациональнее однообразные функции объединять в отдельные интерфейсы.

Возникает закономерный вопрос — как быть, если клиент не знает, какие интерфейсы имеются у объекта? Для получения их перечня нужно использовать базовый интерфейс `IUnknown`, который есть у любого объекта COM. К рассмотрению этого интерфейса мы сейчас и перейдем.

## Интерфейс *IUnknown*

`IUnknown` является базовым интерфейсом COM-объектов. Через этот интерфейс можно получить все остальные интерфейсы, которые поддерживает объект. В нем присутствуют всего три метода, но они играют самую важную роль в функционировании объекта:

- `QueryInterface`
- `AddRef`
- `Release`

Метод `QueryInterface` возвращает указатель на интерфейс объекта по его идентификатору. Если передан идентификатор несуществующего интерфейса, то метод возвратит `Null`.

Кроме предоставления независимого от языка программирования доступа к методам объектов, COM реализует автоматическое управление памятью для COM-объектов, основанное на идее подсчета ссылок на объект. Объект существует, пока его использует хотя бы один клиент. Поэтому любой клиент после создания объекта должен увеличить счетчик ссылок, а после завершения его использования уменьшить счетчик на единицу. Когда счетчик достигнет нулевого значения, COM-объект автоматически будет удален из памяти. Именно для этого предназначены метод `AddRef`, увеличивающий счетчик на единицу, и метод `Release`, уменьшающий его.

Стоит заметить, что, пользуясь функцией `QueryInterface` для получения указателя на интерфейс в среде Delphi, метод запускает процедуру увеличения счетчика ссылок. Вызвать его вручную может потребоваться в том случае, если один клиент попытается передать другому указатель на интерфейс объекта. Тогда без вызова `AddRef` счетчик будет хранить неверные сведения о количестве использующих его клиентов. То же справедливо и для метода `Release`. При выходе переменной, ссылающейся на интерфейс, за область видимости (либо при присвоении ей другого значения) компилятор Delphi генерирует код для вызова метода `Release`, информируя реализацию COM-объекта о том, что ссылка на нее больше не нужна. Поэтому нет надобности постоянно вызывать этот метод, за исключением особых случаев.

## Сервер

Объект работает в составе *сервера COM*. Сервер может быть как динамической библиотекой, так и исполняемым файлом. Объект имеет свои собственные свойства и методы или пользуется данными и службами сервера.

COM-серверы бывают трех типов:

- *внутренний сервер* — обычно это динамические библиотеки, подключаемые к приложению. Функционируют в одном адресном пространстве, поэтому называются еще *in-process server*;
- *локальный сервер* — для такого сервера создается отдельный процесс, работающий на одном компьютере с приложением-клиентом;
- *удаленный сервер* — работает в своем процессе на другом компьютере по отношению к клиенту.

## Библиотека COM

Итак, мы разобрались с тем, как получить указатель на интерфейс — для этого используется метод `QueryInterface`. Но что же делать, если клиенту требуется получить интерфейс объекта, который еще не использовался и, следовательно, может быть и не создан. В этом случае клиенту необходимо обратиться к *библиотеке COM*. Она обеспечивает выполнение базовых

функций и интерфейсов в операционной системе. К ней обращаются посредством специальных функций, имена которых согласно спецификации начинаются с приставки `Co`.

При установке COM-приложения в системный реестр записываются данные обо всех реализуемых им объектах.

- `CLSID` (`Class Identifier`) — идентификатор класса, однозначно идентифицирует класс объекта в системе;
- тип сервера объекта (внутренний, локальный, удаленный):
  - для внутренних и локальных серверов записывается полное имя динамической библиотеки или исполняемого файла;
  - для удаленных — полный сетевой адрес.

Поэтому для получения указателя на требуемый класс и интерфейс необходимо вызвать метод `CoCreateInstance`, передав в качестве параметров CLSID нужного класса, IID интерфейса (Interface IDentifier) и тип требуемого сервера.

Возвращение указателя происходит по следующей схеме.

1. Библиотека через диспетчер управления службами обращается к системному реестру.
2. Находит информацию о сервере по идентификатору класса CLSID.
3. Запускает сервер.
4. Сервер создает экземпляр класса.
5. Объект возвращает библиотеке указатель на запрошенный интерфейс.
6. Библиотека COM передает указатель клиенту.

## Другие элементы COM

В данном разделе мы не рассмотрели некоторые другие базовые элементы COM, однако они не менее важны.

Для запуска экземпляра класса служит специальный объект — *фабрика класса*. С его помощью можно создать один или несколько экземпляров некоторого класса. То есть для каждого класса существует своя фабрика класса.

Разработчик COM-объектов для пользователей создает специальную информацию, описывающую типы объектов, чтобы документировать интерфейсы. Вся информация составляется на языке IDL и объединяется в специальной *библиотеке типов*.

Все вышесказанное может показаться сложным, и тем не менее, это так только на первый взгляд. В следующем разделе мы перейдем непосредственно к программированию и "пощупаем" все своими руками.

# 1.3. Технология COM в среде Delphi

Сейчас мы познакомимся со спецификой реализации COM-приложений в Delphi. На первом этапе изучим способы объявления интерфейсов в собственных приложениях, а затем рассмотрим детали их реализации. После этого узнаем, как обратиться к главному приложению из дополнительных модулей, таких как DLL, с помощью интерфейсов. Наконец, создадим полноценное COM-приложение с несколькими интерфейсами и методами.

## Объявление интерфейсов

В Delphi существует специальное ключевое слово, расширяющее набор `Object Pascal` для поддержки интерфейсов. Этим ключевым словом является слово `interface`.

```
IFirstInterface = interface
  //описание интерфейса
end;
```

Мы уже знаем, что каждый интерфейс имеет имя и уникальный идентификатор GUID. С именем все более или менее понятно: оно придумывается разработчиком и начинается с буквы `I`. В данном случае интерфейс носит имя `IFirstInterface`.

Однако что делать с GUID — глобальным идентификатором? Для глобального идентификатора в Delphi имеется специальный тип `TGUID`:

```
TGUID = packed record
  D1: LongWord;
  D2: Word;
  D3: Word;
  D4: array[0..7] of Byte;
end;
```

В прикладном интерфейсе программирования Windows (WinAPI) существует специальная функция для генерации нового идентификатора — `CoCreateGUID`. Она описана в модуле `ActiveX.pas`.

Для преобразования идентификатора в строку и обратно используются соответствующие функции из модуля ComObj.pas:

```
function GUIDToString(const ClassID: TGUID): string;
function StringToGUID(const S: string): TGUID;
```

В Delphi имеется и более простой способ генерации нового идентификатора. Для вставки нового идентификатора в нужное место необходимо установить туда курсор и нажать комбинацию клавиш <Ctrl>+<Shift>+<G>. Действуя

по этому способу, в описании нашего нового интерфейса поставьте курсор на следующую строку и сгенерируйте идентификатор. Получится результат, подобный приведенному ниже:

```
IFirstInterface = interface
['{34381141-4B31-11D8-8903-0020ED19BE94}']
  //Описания методов интерфейса
end;
```

Теперь можно описывать требуемые методы, которые должен включать описываемый интерфейс. Например, объявим ничего не возвращающий метод `Hello` без параметров:

```
IFirstInterface = interface
['{34381141-4B31-11D8-8903-0020ED19BE94}']
   procedure Hello();
end;
```

Теперь можно приступать к реализации методов.

## Реализация интерфейсов

Реализацией интерфейса в Delphi всегда выступает класс. Поэтому в объявлении класса необходимо указать, какие интерфейсы он реализует.

> ***О реализации***
>
> Возвращаюсь к примеру неофициального определения интерфейсов. Может показаться странным, что нельзя просто написать реализацию в секции `implementation`. Однако в этом случае интерфейс не отличался бы от обычного класса. В ситуации, когда мы являемся клиентом, наша клавиатура — сервером, а наши руки — интерфейсом, реализация интерфейса не находится у нас в руках. Метод нажатия клавиши реализует клавиатура! В ней реализовано то, как именно должны замыкаться контакты, как должен подаваться сигнал в компьютер.

Рассмотрим пример объявления класса, реализующий интерфейс.

```
type
  IFirstInterface = interface
  ['{34381141-4B31-11D8-8903-0020ED19BE94}']
     procedure Hello();
  end;

  TFirstClass = class(TInterfacedObject, IFirstInterface)
    procedure Hello();
  end;
```

Подобное объявление не свидетельствует о наличии множественного наследования в Delphi в том виде, как оно реализовано в C++. Это говорит о том, что в данном классе реализованы все указанные интерфейсы.

Класс должен иметь методы, точно соответствующие по именам и спискам параметров всем методам, объявленным в его заголовке интерфейсов. То есть необходимо написать реализацию всех методов интерфейсов, указанных в объявлении класса, причем имена этих методов должны совпадать с именами методов, указанных в описании интерфейса, а также должны совпадать списки их параметров.

Новый класс является наследником от `TInterfacedObject`, рекомендуется именно его использовать для реализации своих собственных интерфейсов.

Полный код примера объявления и реализации интерфейса расположен в каталоге Chapter1\01_FirstInterface\ на прилагаемом компакт-диске.

Теперь рассмотрим более подробный пример (см. листинг 1.1). Опишем некоторый интерфейс и реализуем его в классе `TFirstClass`, а затем запустим какой-нибудь из его методов. Запуск будет производиться щелчком мыши на кнопке `Button1`.

**Листинг 1.1. Демонстрация работы интерфейса (проект из каталога 02_Simple)**

```
type
  IFirstInterface = interface
  ['{34381141-4B31-11D8-8903-0020ED19BE94}']
     procedure Hello(); stdcall;
  end;

  TFirstClass = class(TInterfacedObject, IFirstInterface)
    procedure Hello(); stdcall;
    destructor Destroy; override;
  end;

  TForm1 = class(TForm)
    Button1: TButton;
    procedure Button1Click(Sender: TObject);
  end;

var
  Form1: TForm1;
```

```
implementation

{$R *.DFM}
//Реализации интерфейса
procedure TFirstClass.Hello(); stdcall;
begin
 ShowMessage('Hello!')
end;

destructor TFirstClass.Destroy;
begin
 ShowMessage('Запустился деструктор!');
end;

//Обработка щелчка мыши на кнопке:
procedure TForm1.Button1Click(Sender: TObject);
var
 localVar: IFirstInterface; //Локальная переменная
begin
  localVar:=TFirstClass.Create;//Приводим класс к интерфейсу
  localVar.Hello();//Вызываем метод
end;
```

Приведенный в листинге 1.1 код может показаться странным в том плане, что мы нигде не увеличиваем и не уменьшаем счетчик ссылок, а также не освобождаем память из-под объекта `localVar`. Однако приведение класса к интерфейсу неявно увеличивает счетчик ссылок на единицу. Кроме того, при выполнении программы появится сообщение о том, что был вызван деструктор. Это происходит потому, что при выходе переменной, ссылающейся на интерфейс, за область видимости (либо при присвоении ей другого значения) компилятор Delphi генерирует код для вызова метода `_Release`, информируя реализацию о том, что ссылка на нее больше не нужна. Таким образом, после выполнения первой строки кода счетчик увеличится на единицу, а при выходе из процедуры уменьшится на единицу, и объект будет удален из памяти.

Рассмотрим теперь код (см. листинг 1.2), который вызовет появление ошибки вследствие неправильной работы с классом, реализующим интерфейс (пример из подкаталога 03_ErrorCode).

**Листинг 1.2. Демонстрация ошибки (пример из каталога 03_ErrorCode)**

```
procedure TForm1.Button1Click(Sender: TObject);
var
 VInt: IFirstInterface;
 VCls: TFirstClass;
begin
 VCls := TFirstClass.Create;
 VInt := VCls;//Передали указатель другой переменной
 VInt.Hello;//Отработала процедура
 VCls.Free;//Удаляем реализацию интерфейса
 //После выполнения этой процедуры будет ошибка
end;
```

При выходе из процедуры возникнет ошибка, т. к. при передаче указателя интерфейсу счетчик увеличится на единицу. После удаления реализации останется указатель на интерфейс, и, соответственно, при выходе переменной за область видимости неявно будет вызван метод `_Release`. Тут и произойдет исключение недействительной операции с указателем, т. к. реализация будет уже удалена.

Поэтому никогда не вызывайте деструктор класса явно — объект будет удален, как только отпадет необходимость в его последней ссылке. Если в данном случае вам необходимо удалить реализацию немедленно, нужно просто присвоить указателю значение `nil`, как показано в листинге 1.3.

**Листинг 1.3. Демонстрация правильного кода**

```
var
 VInt: IFirstInterface;
 VCls: TFirstClass;
begin
 VCls := TFirstClass.Create;
 VInt := VCls;//Передали указатель другой переменной
 VInt.Hello;//Отработала процедура
 VInt:=nil;//После этого неявно будет вызван IUnknown._Release
           //и уничтожена реализация, т. к. отпадет необходимость
           //в последней ссылке
end;
```

## Другие способы реализации интерфейсов

Нередко встречаются ситуации, когда класс должен реализовывать несколько интерфейсов. В этом нет ничего сложного, т. к. необходимо всего лишь, чтобы методы класса совпадали по именам и спискам параметров с методами, объявленными в его заголовке интерфейсов. Но как поступить в случае, когда в нескольких различных интерфейсах присутствуют одноименные методы? Например, в случае, показанном в приведенном ниже фрагменте кода:

```
IFirstInterface = interface
['{34381141-4B31-11D8-8903-0020ED19BE94}']
   procedure Hello();
end;

ISecondInterface = interface
['{34381142-4B31-11D8-8903-0020ED19BE94}']
   procedure Hello();
end;
```

При описании реализации методов интерфейсов в каком-либо классе возникнет переобъявление методов. Выйти из положения можно следующим образом (см. также проект в каталоге 04_TwoInterface):

```
TFirstClass = class(TInterfacedObject, IFirstInterface, ISecondInterface)
     procedure Hello1;
     procedure Hello2;
     procedure IFirstInterface.Hello = Hello1;
     procedure ISecondInterface.Hello = Hello2;
end;
```

Конфликт разрешается явным указанием на то, какой метод служит реализацией соответствующего метода интерфейса.

Если реализация методов должна быть идентичной, то обычно проблему решают, назначая один и тот же метод класса реализацией нескольких методов интерфейсов. То есть поступают следующим образом:

```
TFirstClass = class(TInterfacedObject, IFirstInterface, ISecondInterface)
     procedure Hello;
     procedure IFirstInterface.Hello = Hello;
     procedure ISecondInterface.Hello = Hello;
end;
```

Демонстрационный проект находится в подкаталоге 05_TwoInterface2.

# Интерфейсы и класс *TComponent*

В базовом классе `TComponent` имеется полный набор методов, позволяющих реализовать интерфейс `IUnknown`, хотя сам класс данный интерфейс не реализует. Это позволяет наследникам `TComponent` реализовывать интерфейсы, не заботясь о реализации `IUnknown`.

Методы `TComponent._AddRef` и `TComponent._Release` на этапе выполнения программы не реализуют механизм подсчета ссылок, т. е. в отношении классов-наследников `TComponent`, реализующих интерфейсы, не действует автоматическое управление памятью. Это позволяет запрашивать у них интерфейсы, не опасаясь, что объект будет удален из памяти при выходе переменной за область видимости.

В качестве примера (см. листинг 1.4) рассмотрим аналогичный приведенному в листинге 1.2 код, который работает корректно по сравнению с кодом из проекта 03_ErrorCode.

**Листинг 1.4. Демонстрация корректного кода (проект каталога 06_GoodCode)**

```
type
  IMyInterface = interface
  ['{DE9ADBC0-4CD7-11D8-8903-0020ED19BE94}']
    procedure Hello();
  end;

  TIntClass = class(TComponent, IMyInterface)
    procedure Hello();
  end;

  TForm1 = class(TForm)
    Button1: TButton;
    procedure Button1Click(Sender: TObject);
  end;

var
  Form1: TForm1;

implementation
{$R *.DFM}
```

```
procedure TIntClass.Hello();
begin
  ShowMessage('Hello!');
end;

procedure TForm1.Button1Click(Sender: TObject);
var
 VInt: IMyInterface;
 VCls: TIntClass;
 i: Integer;
begin
 VCls:=TIntClass.Create(Self); //Обратите внимание на параметр Self
 VInt:=VCls;//Передали указатель другой переменной
 VCls.Hello();//Отработала процедура
 i:=VCls._Release;
 ShowMessage(IntToStr(i));//Здесь мы получим сообщение "-1"
 VInt.Hello();//А вот в этом месте уже НЕ будет ошибки!
end;
```

Благодаря классу `TComponent` в методе `Create` появился параметр `AOwner: TComponent`, т. е. "хозяин". Поэтому `VCls` может быть удален из памяти методом `Free` вручную, хотя он также очистит память при уничтожении формы, т. к. теперь она является хозяином данного объекта.

## Создание подключаемых модулей *Plugins* с помощью интерфейсов

Те объекты, которые мы создавали в предыдущих разделах, еще нельзя назвать полноценными COM-объектами, т. к. они не входят в состав COM-сервера, информация о них не записывается в реестр и они не выполняют базовой функции — осуществления обмена данными между приложениями (вспомните, с чего все началось).

Тем не менее, проблемы, которые будут рассмотрены в данном разделе, невозможно решить с помощью обычных классов ООП без применения интерфейсов.

Представим себе такую ситуацию. Существует приложение типа записной книжки, в которой реализованы функции чтения, записи и т. п. Однако вы желаете обеспечить импорт в приложение данных из других форматов.

Соответственно, необходимо реализовать идею подключаемых модулей — plugins ("плагинов"). Модуль знает, как прочитать данные в определенном формате и записывает их в базу данных или в файл основной программы. В этой идее есть один нюанс. Что же получается — нам необходимо предоставлять прямой доступ к данным главной программы? (В нашем случае к базе данных или файлу). Однако вам вряд ли понравится, если ваш коллега напишет подключаемый модуль, который однажды удалит какие-нибудь важные данные. Возможно, во второй раз вы им и не воспользуетесь, да только информация уже будет удалена.

Очевидно, что в подобном случае главная программа должна предоставлять интерфейсы конечным разработчикам, закрывая для доступа детали реализации внутреннего механизма работы программы.

Надеюсь, что многие читатели сталкивались с созданием *динамических библиотек*. Обычно динамическая библиотека (DLL-библиотека) предоставляет функции вызывающему приложению. Теперь же мы попробуем построить модуль и главное приложение так, чтобы и из вспомогательного модуля (!) можно было вызвать некоторые функции главной программы.

Для удобства реализации мы упростим заданную ситуацию и создадим пример, в котором возможно из DLL-библиотеки изменить значение некоторой переменной главной программы, причем реализация определенного метода интерфейса будет контролировать допустимые значения (в данном случае — не допускать ввод значений меньших −100).

Начнем с того, что описание интерфейса (см. листинг 1.5) должно находиться в отдельном модуле. Его мы будем распространять среди конечных разработчиков.

**Листинг 1.5. Модуль описания интерфейса**

```
unit InterfaceUnit;

interface

uses
  Classes;

type
  IPluginInterface = interface
  ['{34381143-4B31-11D8-8903-0020ED19BE94}']
     procedure Hello();
```

```
    function GetLocalVar(): integer;
    procedure SetLocalVar(sInt: integer);
  end;

implementation

end.
```

Этот модуль мы подключим к модулю основной формы главного приложения (см. листинг 1.6), причем укажем там, что класс главной формы реализует требуемый интерфейс (или интерфейсы) и опишем реализацию методов интерфейса, как делали раньше.

**Листинг 1.6. Листинг модуля главного приложения (секции `interface` и `implementation` значительно упрощены)**

```
unit Unit1;

interface

uses
  ..., InterfaceUnit;

type
  TForm1 = class(TForm, IPluginInterface)
  ...
  public
    { Public declarations }
    procedure Hello();
    function GetLocalVar(): integer;
    procedure SetLocalVar(sInt: integer);
  end;

var
  Form1: TForm1;
  ...

implementation
```

```
var
 TestInt: integer;

//Реализация интерфейса
procedure TForm1.Hello();
begin
 ShowMessage('Привет из главного приложения!')
end;

function TForm1.GetLocalVar(): integer;
begin
 Result:=TestInt;
end;

procedure TForm1.SetLocalVar(sInt: integer);
begin
 if sInt >= -100 then
   TestInt:=sInt
 else
   ShowMessage('Нельзя присвоить значение меньше -100!')
end;

...

end;
```

Затем опишем загрузку библиотеки и вызов функций обычным способом (подробнее см. коды основной программы проекта каталога 07_Plugins).

Теперь перейдем к DLL. Строим ее как обычную библиотеку, подключаем в раздел `uses` модуль, в котором описаны необходимые интерфейсы, и создаем процедуры и функции для инициализации плагина и его выгрузки. Рассмотрим этот этап подробнее.

В процедуре инициализации модуля необходимо передавать из главного приложения объект `Application` для того, чтобы получить доступ к главной форме программы. Естественно, основное приложение должно передавать значение своего объекта `Application`. Для удобства будем преобразовывать тип `TApplication` к типу `Integer`.

После проделанных действий мы можем обратиться к главной форме приложения и запросить у нее указатель на нужный интерфейс:

```
var
  Plg: IPluginInterface;

procedure InitPlugin(App: integer); StdCall;
begin
  ...
  Application := TApplication(App);
  Application.MainForm.GetInterface(IPluginInterface, Plg);
end;
```

В данном примере указатель на интерфейс получается посредством вызова функции:

```
function GetInterface(const IID: TGUID; out Obj): Boolean;
```

Так как `MainForm` является свойством класса `TApplication`, то метод `QueryInterface` из секции `protected` класса `TCustomForm` для него не поддерживается. Однако ничто не мешает воспользоваться методом `GetInterface`, реализованным еще в классе `TObject` в секции `published`.

На этом можно поставить точку, т. к. через указатель на интерфейс, хранящийся в переменной `Plg`, можно получить доступ ко всем методам, которые поддерживает данный интерфейс.

Запустите основную программу. Попытайтесь изменить значение переменной `TestInt` и получить его в форме, вызванной из DLL. Затем, наоборот, измените значение переменной в дополнительном модуле и посмотрите, чему будет оно равно в главном. Значения всегда будут совпадать, т. к. мы работаем с одной переменной. Обратите внимание на то, что невозможно задать число меньшее −100. Контроль ввода происходит в реализации главной программы, т. е. от конечного разработчика ничего не зависит. Тем самым мы скрыли детали реализации нашего приложения.

Подобным образом можно реализовать запись необходимых данных в базу данных и файл. Необходимо всего лишь, чтобы в главном приложении были соответствующим образом реализованы методы интерфейсов.

## Первый COM-объект

Настало время создать наш собственный COM-объект. Включим его в состав динамической библиотеки.

Пусть наш сервер называется `FirstServer`, объект — `FirstCOM`. Объект будет включать один-единственный интерфейс — `IFirstCOM` с методом `GetSqr`.

Этот метод будет возвращать квадрат переданного числового значения. Конечно, для нахождения квадрата числа совершенно необязательно пользоваться моделью COM, однако для нас лучше показать общую схему на простом примере.

Для начала необходимо создать ActiveX Library — будущий сервер. В меню **File | New** (Файл | Новый) на вкладке **ActiveX** следует выбрать именно этот тип приложения, т. к. в данном случае Delphi сгенерирует необходимый код для вызова служебных функций. Перед вами откроется заготовка библиотеки — это и будет наш сервер. Теперь добавим к нему COM-объект. Снова на той же вкладке **ActiveX** следует выбрать пункт **COM Object** (COM-объект). Появится диалоговое окно примерно такого вида, как показано на рис. 1.1.

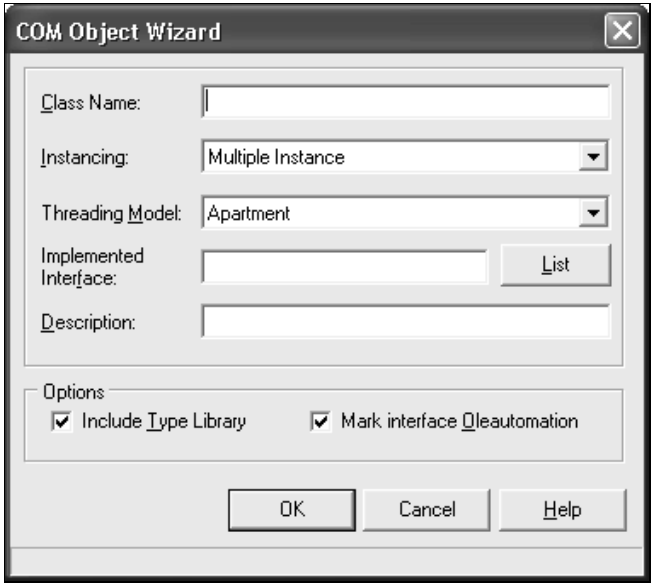


**Рис. 1.1.** Диалоговое окно мастера создания COM-объектов

Верхнее поле редактирование **Class Name** (Имя класса) определяет имя нового класса.

Раскрывающийся список **Instancing** (Типы объекта) определяет способ создания объекта:

- **Internal** (Внутренний объект) — объект используется в процессе приложения. В этом случае никакие данные не записываются в реестр, т. е. объект не будет доступен внешним процессам;