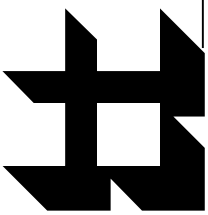

ГЛАВА 16

Технология COM

COM — это сокращенное название разработанной корпорацией Microsoft модели объектных компонентов (Component Object Model). Эта модель определяет двоичный стандарт объектов в Windows. Объекты COM скомпонованы в исполняемые файлы (обычно с расширением dll или exe), что позволяет приложениям Windows в процессе выполнения просто загружать в память код такого объекта без какой-либо дополнительной трансляции и затем работать с ним как с "собственным" объектом, присваивая значения его свойствам и вызывая методы.

Существует соглашение, согласно которого приложение, использующее COM-объект, называется *клиентом*, а сам COM-объект — *сервером*. Часто бывает так, что один и тот же объект одновременно является и клиентом, и сервером. Например, ваше приложение всегда использует COM-сервер с именем FoxApplication из библиотеки vfp9r.dll; в то же время этот сервер постоянно связывается с другими COM-серверами и является по отношению к ним клиентом.

К семейству COM относится множество различных широко применяемых компонентов Windows, таких как COM-серверы из Windows Script Host или управляющие элементы ActiveX. Многие полнофункциональные приложения так же реализованы, как "большие COM"; в частности, Microsoft Excel, Microsoft Word и, конечно, Visual FoxPro являются COM-серверами.

Компоненты COM могут быть созданы практически на любом из распространенных языков программирования, поддерживаемых в Windows, и могут быть использованы во множестве приложений, поддерживающих технологию COM. Именно применение этой технологии позволяет вам из приложения, написанного на Visual FoxPro, формировать отчеты в Microsoft Excel, работая с ним так, как если бы он был одним из "собственных" объектов Visual FoxPro.

Важнейшим свойством COM является *независимость от местоположения* (Location Transparency). Независимость от местоположения означает, что пользователь компонента не должен знать, где находится этот компонент — на том же самом компьютере, на котором выполняется запущенное пользователем приложение, или на компью-

тере, расположенном за сотни километров. Если ваше приложение реализуется как "большой COM", то вы фактически разбиваете его на несколько согласованно работающих, но самостоятельных частей. Причем совершенно не существенно, работают ли эти части на одной машине или на нескольких. В случае внесения каких-то изменений проще переписать и отладить отдельный компонент (причем не важно, на каком языке этот компонент написан), что, безусловно, проще переписывания и последующего (в полном объеме) тестирования всего приложения. Это делает COM очень удобной архитектурной платформой для проектов, масштаб которых заранее не очень понятен и впоследствии может измениться.

В этой главе вы прикоснетесь к основам технологии COM. Вы узнаете, чем отличаются COM-классы от обычных классов Visual FoxPro, а также научитесь создавать различные COM-серверы.

Введение в технологию COM

Вы можете создать COM-класс как программно, вручную написав его код, так и в Конструкторе классов. Все зависит от того, что вы хотите в результате получить.

Если ваша цель — создать COM-сервер, поддерживающий интерфейс пользователя (формы, диалоговые окна и т. п.), то вам лучше воспользоваться Конструктором классов. Конечно, в Конструкторе классов вы можете создавать и не визуальные COM-классы, ориентированные на работу с данными (как известно, Visual FoxPro ориентирован именно на работу с данными), но лучше такие COM-классы создавать программно. В частности, только программно можно создать COM-класс из базового класса `Session`; этот класс специально разработан для применения в COM и позволяет реализовать личную сессию данных (private data session) для каждого потока.

Мы рассмотрим в этой главе как программный, так и визуальный способ создания COM-сервера. Но сначала немного теории.

Процессы и потоки

Процессом (process) называется область памяти, в которую загружен выполняющийся экземпляр приложения и все необходимые для его успешного выполнения ресурсы: файлы, растровые изображения, динамически загружаемые библиотеки и различные двоичные исполняемые модули, связанные с приложением, в том числе и COM-объекты. Запрещается непосредственный доступ к памяти, занимаемой процессом, из других процессов.

Если COM-сервер загружается в ту же область памяти, в которой выполняется клиент, то говорят, что такой сервер выполняется *in-process*, т. е. *в процессе* клиента. Основным преимуществом связи внутри процесса является высокое быстродействие, недостатком — низкая надежность: при возникновении ошибки на сервере выполнение клиента так же будет прекращено. Все серверы, выполняющиеся in-process, компонуются как DLL-модули.

Если COM-сервер создает собственный процесс для выполнения, то говорят, что он выполняется *out-of-process*, т. е. *вне процесса* клиента. Клиент и out-of-process сервер могут выполняться на одном компьютере (локальная связь) или на разных компьютерах (удаленная связь). При возникновении ошибки в сервере out-of-process выполнение клиента может быть продолжено. Недостаток такого сервера — невысокое быстродействие, потому что он вынужден использовать специальный механизм для передачи информации между процессами. Все серверы out-of-process компонуются как EXE-модули.

Каждый процесс имеет как минимум один первичный поток, который является точкой входа в программу и запускается сразу же после загрузки приложения.

Потоком (thread) называется последовательность действий внутри процесса. Сам процесс ничего не исполняет, он просто служит контейнером для потоков. Потоки всегда создаются в контексте какого-либо процесса, и вся их жизнь проходит только в его границах. На практике это означает, что потоки исполняют код и манипулируют данными в адресном пространстве процесса. Поэтому если несколько потоков выполняются в контексте одного процесса, то они делят одно общее адресное пространство. Потоки могут исполнять один и тот же код и манипулировать одними и теми же данными. Большинство приложений обходится единственным, первичным потоком. Однако процессы могут создавать дополнительные потоки, что позволяет им эффективнее выполнять свою работу. Например, для операции типа печати, требующей значительного времени, приложение могло бы создать поток, который будет выполняться в фоновом режиме. Первичный поток в это время сможет обрабатывать запросы пользователей.

На многопроцессорных компьютерах приложения, использующие несколько потоков, могут выполняться весьма эффективно, поскольку потоки связываются операционной системой с отдельными процессорами. На компьютерах с одним процессором многопоточность — это часто иллюзия, создаваемая операционной системой. Процессор выделяет каждому потоку для выполнения некоторый интервал времени, определяемый приоритетом потока. Когда выделенный интервал заканчивается, поток приостанавливается для того, чтобы предоставить возможность выполняться другому потоку. При переключении потоков операционная система вынуждена сохранять реквизиты останавливаемого потока и восстанавливать реквизиты запускаемого потока, что также требует процессорного времени. Поэтому на компьютерах с одним процессором эффективность выполнения многопоточного приложения может оказаться ниже, чем однопоточного.

Если ваш COM-сервер может создавать только один поток, то тогда для каждого клиента будет загружаться свой экземпляр COM-сервера. Если сервер может создать несколько потоков, то тогда объект — экземпляр COM-сервера будет создаваться только первым клиентом. Каждый последующий клиент будет запускать связанный с ним новый поток внутри процесса COM-сервера.

Возможны и гораздо более сложные ситуации, когда многопотоковыми являются и клиент, и сервер. Более того, такие клиенты и серверы могут выполняться на различных компьютерах в сети. Здесь на первый план выходят вопросы целостности и за-

щиты данных, для решения которых Windows предлагает специальные средства, например, реализованные как Служба COM+-компонентов, рассмотрение которых выходит за рамки данной книги.

Идентификаторы

Каждый COM-класс зарегистрирован в реестре Windows. Именно использование реестра позволяет реализовать концепцию независимости компонента от его расположения. Вашему приложению нет необходимости знать, в каком файле, в какой папке и на каком томе располагается загрузочный модуль COM-компонента. Для локализации компонента достаточно найти в реестре его программный идентификатор (ProgID), затем извлечь всю связанную с этим идентификатором информацию, в том числе и определить местоположение загрузочного файла.

Программный идентификатор — это строка, состоящая из разделенных точкой имени сервера и имени класса, например:

```
Excel.Application
```

Один сервер может включать в себя множество COM-классов. Так, Microsoft Excel включает в себя классы со следующими ProgID:

```
Excel.WorkBook
Excel.WorkSheet
Excel.Chart
```

и т. д.

Помимо программного идентификатора, каждый COM-класс обладает уникальным идентификатором (Class Identifier, CLSID).

CLSID относится к семейству глобальных универсальных идентификаторов (Globally Unique Identifier, GUID). GUID представляет собой двоичное 128-разрядное число, вычисляемое по специальному алгоритму. Этот алгоритм для большей надежности использует ряд действительно случайных значений. Например, в подсчете используется показание внутреннего таймера машины, некоторые параметры BIOS, уникальный идентификатор сетевой карты. Математически доказано, что вероятность получения двух одинаковых значений GUID в течение ближайших веков практически равна нулю.

Почему длина GUID равна 128 битам? Да просто для удобства обработки и обращения. Быстрее и проще работать с переменной, имеющей длину, кратную длине слова процессора.

Для чего нужен уникальный идентификатор класса? Для обеспечения уникальности COM-класса. Невозможно требовать уникальности для ProgID; вряд ли возможно добиться того, чтобы все программисты мира именовали свои COM-классы так, чтобы их имена не совпадали. А идентификатор класса всегда уникален.

Каждый COM-класс определяется только одним, принадлежащим только этому классу, глобальным идентификатором ClsId.

Для отображения уникальных идентификаторов разработан специальный формат. Например, CLSID для Excel.Application (версии 11) выглядит так:

```
{00024500-0000-0000-C000-000000000046}
```

В формате GUID могут применяться только латинские буквы, цифры и дефис. Регистр символов не имеет значения.

Когда вы создаете объект — экземпляр COM-объекта, например, при помощи функции `CREATEOBJECT`, то вы сообщаете Windows ProgID этого класса:

```
O = CREATEOBJECT("Excel.Application")
```

Получив программный идентификатор класса, Windows просматривает реестр и находит в нем всю необходимую информацию для создания этого объекта, включая его расположение, тип, способ запуска и многое другое.

Информация о COM-объектах в реестре Windows

Реестр организован как иерархическая база данных, в которой хранится вся информация, необходимая Windows для работы. Подробно структура реестра и работа с ним рассматривается в *главе 20*, здесь же мы познакомимся только с теми его разделами, которые предназначены для хранения сведений о COM-объектах.

Посмотреть содержимое Реестра можно, вызвав **Редактор реестра** (regedit.exe) командной строки Windows. Окно **Редактор реестра** показано на рис. 16.1.

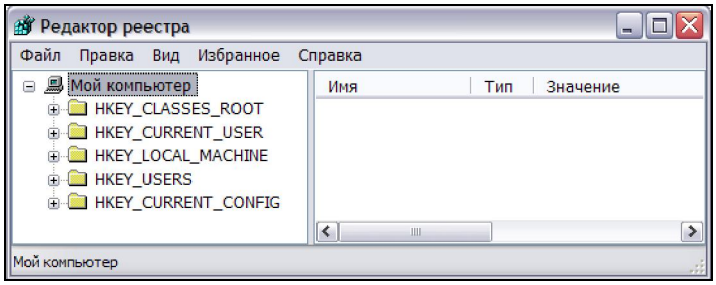

Рис. 16.1. Окно Редактора реестра Windows

В левой части окна располагаются узлы реестра, а в расположенной справа области выводятся различные значения.

Узлы реестра обычно называют разделами или ключами. Реестр имеет несколько главных ключей, имеющих строго определенные имена, которые не изменяются от версии к версии Windows. Каждый ключ открывает раздел реестра и может содержать произвольное количество вложенных ключей. Любой ключ может иметь одно

или несколько значений (а может и не иметь никаких значений — наличие или отсутствие ключа так же является информацией).

Вся информация о COM-классах находится в узле HKEY_CLASSES_ROOT. Этот узел имеет множество вложенных ключей; часть этих ключей имеет имена, совпадающие с программными идентификаторами (ProgID) зарегистрированных COM-компонентов.

Откройте узел HKEY_CLASSES_ROOT и просмотрите раздел до появления ключа Excel.Application. Откройте этот ключ. Появятся два вложенных ключа: CLSID и CurVer. Щелкните по ключу CLSID. В правой части окна реестра вы увидите значение ClsID класса (рис. 16.2).

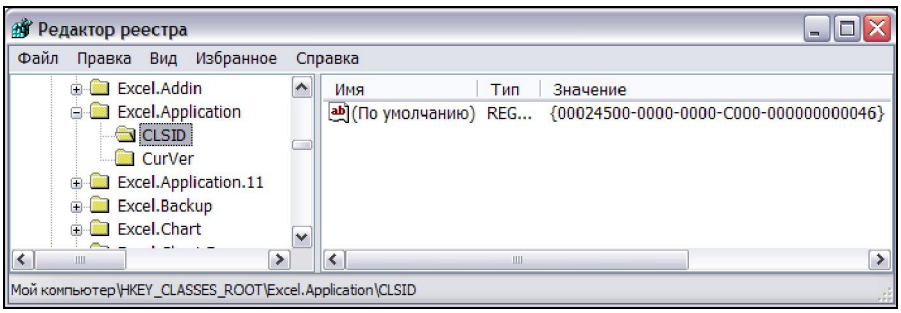

Рис. 16.2. Ключ CLSID для Excel.Application в окне редактора реестра

Когда ваше приложение вызывает функцию для создания COM-объекта, указав ей ProgID, то эта функция ищет в разделе HKEY_CLASSES_ROOT соответствующий этому ProgID ключ для того, чтобы узнать ClsID класса. Затем функция ищет новый ключ с именем, совпадающим с полученным значением ClsID, в разделе HKEY_CLASSES_ROOT/CLSID. На рис. 16.3 показана область Реестра для ключа {00024500-0000-0000-C000-000000000046}.

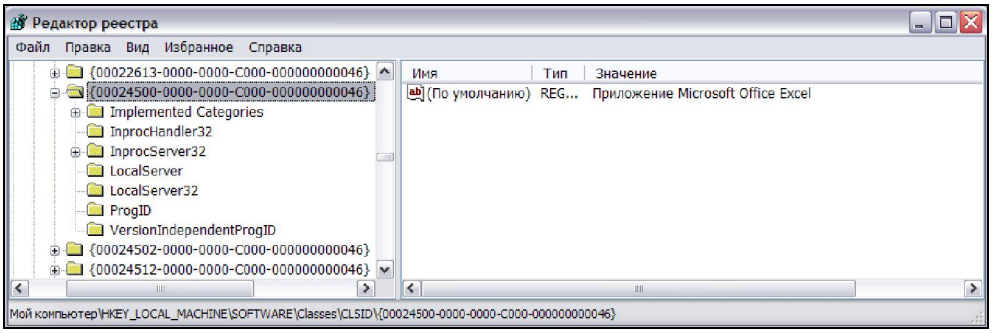

Рис. 16.3. Окно редактора реестра для ключа
HKEY_CLASSES_ROOT/CLSID/{00024500-0000-0000-C000-000000000046}

Если функции, создающей COM-объект, передается значение CLSID, то она сразу обращается в раздел HKEY_CLASSES_ROOT/CLSID реестра.

Как видите, это именно то место реестра, где хранится вся основная информация о COM-объекте. В частности, ключ LocalServer32 хранит путь и имя файла, который будет загружен в память при создании объекта (рис. 16.4).

Ну, хорошо, скажете вы. Объект мы создали и в память процесса загрузили. Но дальше что? Каким образом можно обратиться к этому объекту?

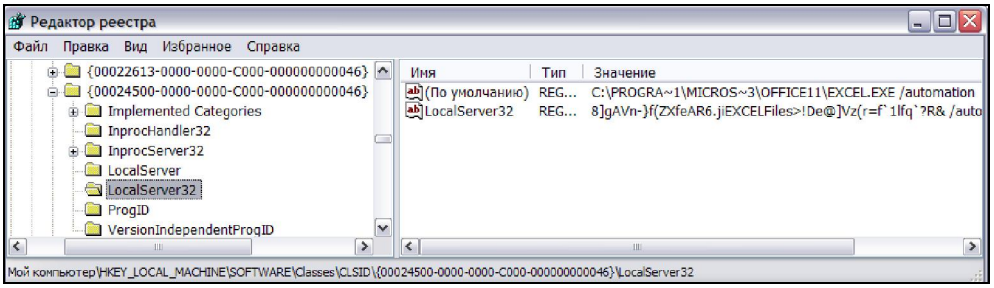

Рис. 16.4. Значение ключа LocalServer32 для объекта с ProgID Excel.Application

Интерфейсы

Любой класс предоставляет миру набор общедоступных (public) методов и свойств, которые образуют *открытый интерфейс* класса.

В общем случае интерфейс можно представить как специальную таблицу, создаваемую на этапе компиляции, содержащую имена и адреса свойств и методов объекта, и некоего указателя на эту таблицу. При обращении к объекту программа использует интерфейс для получения адреса метода или свойства по его имени.

Совершенно иначе дело обстоит при работе с двоичными компонентами. Если сам сервер "знает" о том, какие свойства и методы экспонируются его объектами и по каким адресам внутри него они располагаются, то клиент об этом не имеет никакого представления — ведь клиент и сервер могут выполняться в разных процессах и даже на разных компьютерах! Вот если бы на этапе компиляции или выполнения каким-то образом можно было бы получить информацию о методах COM-класса...

Для решения этой проблемы и придуманы COM-интерфейсы.

В отличие от открытого интерфейса класса, экспонирующего реально существующие свойства и методы, COM-интерфейс является абстракцией, потому что он предоставляет только *информацию* об экспонируемых им методах, но никак не сами эти методы. Сопоставление информации о методе с его реальным адресом внутри двоичного модуля объекта происходит только на этапе создания объекта (его загрузки в память компьютера), или даже только при непосредственном вызове этого метода.

Для того чтобы лучше понять назначение интерфейсов, нам понадобится выйти за рамки Visual FoxPro и углубиться (не очень сильно) в основы C++, чтобы познакомиться с такими понятиями, как чисто виртуальные функции и абстрактные классы.

Чисто виртуальной называется функция, не имеющая никакой реализации; т. е. в коде класса объявлен только ее прототип (понятие "прототипа функции" в Visual FoxPro отсутствует, а о прототипах функций C++ вы узнаете в *главе 19*). Прототип всего лишь указывает компилятору C++, что такая функция может появиться; прототип функции вовсе не есть факт существования такой функции.

Класс, содержащий чисто виртуальные функции, называется абстрактным классом. Из абстрактного класса невозможно создать объект, но от него можно наследовать. Когда программист C++ создает новый класс, являющийся потомком абстрактного класса, то он должен *перегрузить* все чисто виртуальные функции, переопределив реализацию для каждой из них (проще говоря, в классе-потомке *нужно написать код* этой функции). Если этого не сделать, то класс-потомок так же будет абстрактным.

Таким образом, если дочерний класс *реализует* (т. е. содержит коды, или *тела* функций) все объявленные в родительском классе чисто виртуальные функции (а так же, возможно, добавляет собственные функции), то у вас появляется возможность создавать объекты — экземпляры этого класса. Смысл использования чисто виртуальных функций можно выразить так: абстрактный родительский класс указывает всем своим потомкам, *что* они обязаны сделать. А вот *как* сделать — это решают сами классы-потомки, причем каждый потомок может иметь свою собственную реализацию виртуальной функции. Такое поведение классов-потомков называется *полиморфизмом*.

COM-интерфейсы — это не что иное, как абстрактные классы, содержащие только чисто виртуальные функции. Как и любые другие классы, интерфейсы могут наследоваться, при этом класс-потомок может добавлять новые чисто виртуальные функции в дополнение к унаследованным от родительского абстрактного класса.

Имена интерфейсов

Имя интерфейса обычно определяется как имя класса, которому предшествует литера "I". Например, если вы создаете COM-сервер из класса `myclass`, то интерфейс этого класса будет иметь имя `Imyclass`. Помимо имени, каждый интерфейс имеет уникальный идентификатор (Interface Identifier, IID), относящийся к семейству глобальных идентификаторов.

Каждый интерфейс — уникален; он экспонирует уникальный набор методов. Никто, нигде и никогда не имеет права менять структуру уже существующего и опубликованного интерфейса.

Программист, живущий за несколько тысяч километров от вас, создавая COM-класс, наследуемый от одного из опубликованных интерфейсов, должен быть абсолютно уверен в том, что ему будут предоставлены именно те методы, которые он ожидает. Более того, он ожидает, что эти методы будут принимать и возвращать именно те параметры именно того типа, как это определено в интерфейсе.

Интерфейс IUnknown

На вершине иерархии COM-интерфейсов находится интерфейс с именем IUnknown. Это имя означает примерно следующее: "Интерфейс Неизвестно Кто". IID интерфейса IUnknown равен {00000000-0000-0000-C000-000000000046}. Все интерфейсы в COM наследуются от IUnknown.

Интерфейс IUnknown обеспечивает каждый объект COM двумя функциональными возможностями. Объявленный в нем метод `QueryInterface` должен возвращать клиенту указатель на запрашиваемый интерфейс, реализуемый объектом, а методы `AddRef` и `Release` управляют временем жизни объекта.

Каждый COM-объект хранит внутренний "счетчик ссылок", представляющий количество указателей интерфейсов, используемых в данный момент. Когда клиент создает объект и запрашивает указатель на интерфейс, метод `AddRef` должен увеличивать значение счетчика на единицу. Когда клиент освобождает интерфейс (уничтожает объект), то он должен вызвать метод `Release` для уменьшения значения счетчика на единицу. Если при этом выяснится, что значение счетчика равно нулю, объект должен "самоликвидироваться".

Раннее и позднее связывание в COM

COM-классы могут наследоваться как непосредственно от интерфейса IUnknown, так и от любого производного от IUnknown интерфейса. При создании COM-класса определяется его собственный интерфейс, в котором реализуются унаследованные чисто виртуальные методы COM-интерфейса и добавляются новые методы и свойства, реализующие функциональность объекта (в Visual FoxPro вы определяете только собственные свойства и методы класса; все необходимые атрибуты типа CLSID, интерфейсов и иных типов данных генерируются автоматически). При компоновке COM-сервера в двоичный модуль генерируется специальный блок кода, называемый библиотекой типов (Visual FoxPro включает этот код в исполняемый модуль и, кроме этого, создает отдельный файл с именем, совпадающим с именем модуля, и расширением tlb). Библиотека типов предоставляет структурированную информацию о всех типах COM-объекта (классах, интерфейсах, методах, передаваемых и возвращаемых параметрах).

vTable-интерфейсы

Если клиент на этапе компиляции имеет возможность получить сведения о типах сервера (всех этих CLSID, IID и прочих GUID, которые опубликованы в библиотеке типов), то он формирует таблицу виртуальных функций vTable, включающую имена всех экспонируемых интерфейсом методов, и указатель vPTR на эту таблицу (т. е. свой внутренний интерфейс). Во время выполнения при создании объекта — экземпляра COM-класса эта таблица заполняется фактическими адресами методов внутри объекта, и клиент вызывает эти методы точно так же, как если бы он вызывал методы "собственного" объекта.

Взаимодействие между клиентом и сервером на основе vTable-интерфейсов в терминах COM называется *ранним связыванием* (Early Binding). К достоинствам раннего связывания можно отнести высокое быстродействие. К недостаткам — необходимость существования COM-сервера в момент компоновки клиента. Кроме того, если по каким-либо причинам версия COM-сервера изменится, что повлечет за собой изменение всех CLSID и IID, то это потребует повторной компоновки клиента.

Принцип реализации vTable-интерфейсов показан на рис. 16.5.

В Visual FoxPro раннее связывание можно реализовать при помощи функции `CREATEOBJECTEX()`. Вот как выглядит вызов этой функции для установки раннего связывания с Microsoft Excel (версии 11):

```
oExcel = CREATEOBJECTEX("{00024500-0000-0000-C000-000000000046}", "", ;
                        "{000208D5-0000-0000-C000-000000000046}")
```

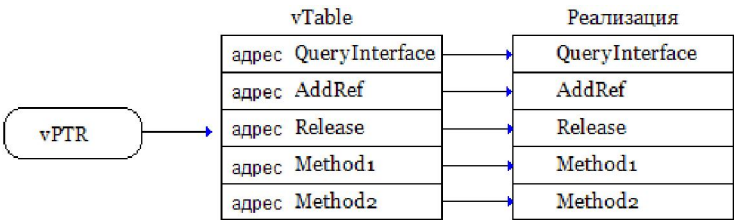

Рис. 16.5. Реализация vTable-интерфейса

Первым параметром функции может быть ProgID или CLSID класса. В показанном выше варианте вызова функции использован идентификатор класса. Если вы хотите передать функции ProgID, то сделать это можно так:

```
oExcel = CREATEOBJECTEX("Excel.Application", "", ;
                        "{000208D5-0000-0000-C000-000000000046}")
```

Второй параметр, передаваемый функции, определяет имя компьютера в сети. Так как мы создаем объект на локальном компьютере, то этот параметр игнорируется.

Третий передаваемый функции параметр — IID интерфейса _Application. Если вы не укажете IID, то будет использован интерфейс "по умолчанию", который функция, создающая объект, найдет в библиотеке типов. При компиляции в приложении-клиенте будет создана таблица виртуальных функций, которая при фактическом создании COM-объекта будет заполнена адресами его методов.

Интерфейс IDispath

Этот интерфейс разработан для использования в тех случаях, когда клиент на этапе компиляции не имеет никакой информации о сервере. Некоторые COM-совместимые языки, в частности, языки для написания скриптов — VBScript и JScript, так же не обеспечивают доступ к интерфейсам vTable. В момент выполнения запроса на соеди-

нение с COM-объектом такой клиент не имеет никакого представления о существовании этого объекта. Клиент как бы надеется, что некий объект с именем "X" имеет такую функцию "Y", которая принимает "столько-то таких-то параметров". Такой способ создания COM-объекта называется *поздним связыванием* (Late Binding), потому что клиент "понимает" функциональность объекта уже в процессе выполнения программы, а не на стадии компиляции. Специально для возможности реализации позднего связывания в COM был введен весьма специфический интерфейс с именем IDispath, наследуемый непосредственно от IUnknown. Его IID равен {00020400-0000-0000-C000-000000000046}. Помимо методов, унаследованных от IUnknown, IDispath определяет еще четыре метода:

- `GetIDsOfNames;`
- `Invoke;`
- `GetTypeInfoCount;`
- `GetTypeInfo.`

При использовании позднего связывания таблица виртуальных функций на стороне клиента содержит описание только тех методов, которые экспонируются интерфейсом IDispath. Клиент не имеет никакого представления о том, какие еще методы экспонирует интерфейс COM-объекта, но он абсолютно уверен в том, что среди них есть семь методов, унаследованных от интерфейса IDispath.

Методы `GetTypeInfo` и `GetTypeInfoCount` интерфейса IDispath позволяют клиенту определить, поддерживает ли объект какую-нибудь информацию о типе (например, библиотеку типов) и если да, то сообщают клиенту эту информацию. Методы `GetIDsOfNames` и `Invoke` предназначены для вызова методов объекта, передачи им параметров и возвращению результата.

Механизм работы интерфейса IDispath показан на рис. 16.6.

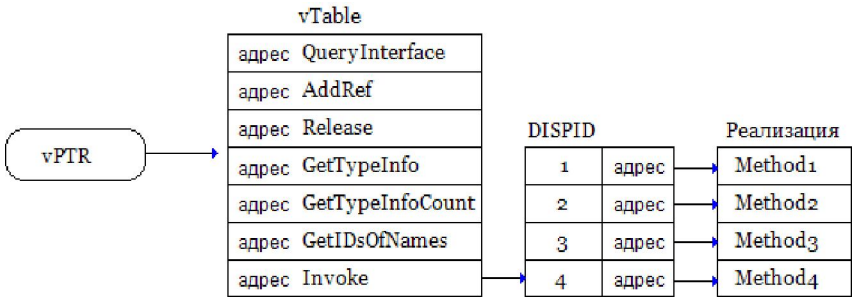

Рис. 16.6. Механизм работы интерфейса IDispath

В процессе выполнения после создания объекта — экземпляра COM-класса таблица виртуальных функций заполняется фактическими адресами методов интерфейса IDispath, реализованными в объекте (напомним, что их всего семь).

В COM-классе, наследуемом от интерфейса IDispath, для каждого экспонируемого метода создается целочисленный идентификатор диспетчеризации DISPID, т. е. все методы класса нумеруются. Когда клиент запрашивает метод объекта, то сначала вызывается метод `GetIDsOfNames`, унаследованный от IDispath. Реализация (код) этого метода содержит перечень наименований методов класса и их DISPID. Получив имя вызываемого метода, `GetIDsOfNames` возвращает значение его идентификатора диспетчеризации. Затем вызывается метод `Invoke`, который реализован как большой условный оператор, где каждое условие соотнесено со значением DISPID; при обнаружении нужного значения DISPID вызывается соответствующий метод класса.

Из-за своей сложности интерфейс IDispath не обеспечивает такого высокого быстродействия, как vTable-интерфейсы, поскольку при обращении к методу объекта дополнительно выполняются вызовы `GetIDsOfNames` и `Invoke`, но зато он обеспечивает очень высокую надежность и устойчивость. И, самое главное, вы можете создавать все новые и новые версии COM-сервера (при неизменном ProgID) без необходимости повторной компиляции и сборки клиента!

Позднее связывание получило настолько широкое распространение, что стало именоваться специальным термином *Автоматизация* (Automation), а COM-серверы, реализующие позднее связывание, называются *серверами автоматизации*.

Позднее связывание при создании объекта реализуется в Visual FoxPro функциями `CREATEOBJECT()` и `NEWOBJECT()`. Эти функции создают объекты по значению ProgID. Вот как, например, можно создать объект Microsoft Excel Application из Visual FoxPro при помощи функции `CREATEOBJECT()`:

```
oExcel = CREATEOBJECT("Excel.Application")
```

Дуальные интерфейсы

Существует еще один способ организации интерфейсов COM-объекта — это дуальные интерфейсы (dual interfaces). Поддерживающие дуальный интерфейс COM-объекты предоставляют как IDispath, так и набор vTable-интерфейсов. В этом случае клиент сам может выбрать тип связывания.

Среда Visual FoxPro и поддержка COM

Команда `SET OLEOBJECT` разрешает или запрещает использование COM в вашем приложении. Если вы уверены, что не будете работать с COM-объектами, и при этом хотите сберечь время и ресурсы, избавив систему от необходимости просматривать реестр Windows, то выполните эту команду с параметром `OFF`. Такая настройка не препятствует работе с ActiveX-компонентами, поскольку Visual FoxPro автоматически загружает средства их поддержки перед тем, как открыть содержащую такие объекты форму или иной объект-контейнер. Поддержка COM по умолчанию включена.

Встроенная функция `COMCLASSINFO` позволяет получить информацию о COM-объекте. Вот ее синтаксис:

```
cType = COMCLASSINFO(oObject [, nInfoType])
```

Первый параметр функции, *oObject*, представляет собой ссылку на COM-объект. Второй параметр, *nInfoType*, — число, определяющее, какая информация должна быть извлечена. Возможные значения этого параметра приведены в табл. 16.1.

Таблица 16.1. *Информация, возвращаемая функцией COMCLASSINFO*

Значение nInfoType	**Возвращаемая информация**
1	ProgID объекта. Например, на вашем компьютере может быть установлено несколько версий Visual FoxPro. Так, для девятой версии функция возвратит следующее значение: VisualFoxPro.Application.9
2	ProgID, не зависящий от версии объекта (VersionIndependentProgId)
3	Оригинальное имя объекта
4	ClsID объекта
5	Тип объекта

Ниже приведена возвращаемая функцией COMCLASSINFO информация об объекте VisualFoxPro.Application, созданном функцией CREATEOBJECT, для различных значений параметра *nInfoType*:

```
1:  VisualFoxpro.Application.9
2:  VisualFoxpro.Application
3:  Microsoft Visual FoxPro Application 9.0
4:  {002d2b10-c1fa-4193-b134-d86eaecc5250}
5:  3
```

В табл. 16.2 приведены возвращаемые функций значения в случае, когда параметр *nInfoType* равен пяти.

Таблица 16.2. *Значения, возвращаемые функцией COMCLASSSINFO, при nInfoType=5*

Возвращаемое значение	**Описание**
1	Это объект Visual FoxPro
2	Это компонент (управляющий элемент) ActiveX
3	Это COM-сервер
4	Это OLEBound-объект (внедренный в поле типа General таблицы)

Программное создание COM-сервера

В этом разделе мы рассмотрим процесс программного создания и настройки COM-сервера.

Создайте новый проект с именем vfpserver, добавьте в него новый программный файл и введите в него код из листинга 16.1.

```
DEFINE CLASS MyServer AS Session OLEPUBLIC
   PROCEDURE DoCmd(cCmd)
      &cCmd
   ENDPROC
   FUNCTION DoExec(cExpr)
      RETURN &cExpr
   ENDFUNC
ENDDEFINE
```

Как видите, объявление COM-класса отличается от объявления обычного класса Visual FoxPro только наличием в команде `DEFINE` предложения `OLEPUBLIC`.

Несмотря на кажущуюся простоту, это очень мощный сервер. Метод `DoCmd` позволяет выполнить практически любую команду Visual FoxPro, а метод `DoExec` выполняет любое допустимое выражение Visual FoxPro и возвращает результат приложению-клиенту.

Убедитесь, что в проекте больше нет никаких подключаемых компонентов (форм, отчетов и прочее — иначе это приведет к ошибке) и нажмите на кнопку **Build...**; в окне **Build Options** (рис. 16.7) выберите **Win32 executable / COM server (exe)**.

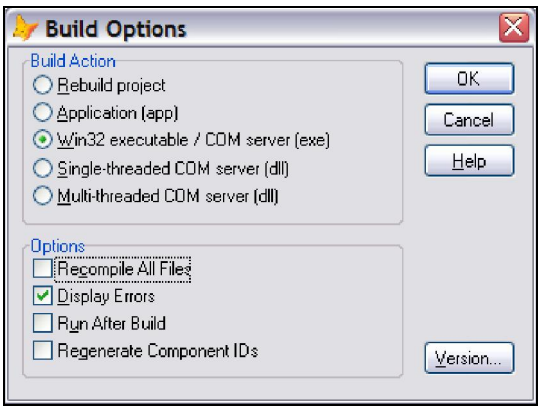

Рис. 16.7. Диалоговое окно **Build Options**

В отличие от обычного проекта, при сборке COM-сервера в случае возникновения ошибки вы не получите подробного сообщения об ее характере и месте расположения; на экран будет выведено диалоговое окно, информирующее вас только о том, что при сборке сервера имела место ошибка. Тем не менее будет сформирован файл с информацией об ошибках (с расширением err). Для того чтобы содержимое этого файла при обнаружении ошибки выводилось на экран, установите флажок **Display Errors**.

Нажмите на кнопку **OK** для создания EXE-модуля.

Теперь необходимо выполнить дополнительные настройки для того, чтобы только что созданный сервер стал полноценным COM. Для этого в главном меню Visual FoxPro выберите пункт **Project** и в выпавшем меню — пункт **Project Info**. В появившемся диалоговом окне **Project Information — Vfpserver** перейдите на вкладку **Servers** (рис. 16.8).

Сейчас нас интересует установка только одного поля — **Instancing**. Значение этого поля выбирается из списка, который содержит следующие варианты:

- **Single Use** — каждое приложение-клиент должно создавать собственный экземпляр COM-сервера. Используется для серверов в виде EXE-модулей.
- **Not Creatable** — сервер может быть использован только в среде Visual FoxPro (т. е. будет невозможно обратиться к вашему COM-объекту из приложений Visual Basic, Delphi и т. п.).
- **Multiple Instancing** — несколько клиентов могут использовать один и тот же объект сервера. Эта установка обычно выбирается в том случае, когда вы создаете многопотоковый сервер в виде DLL-модуля.

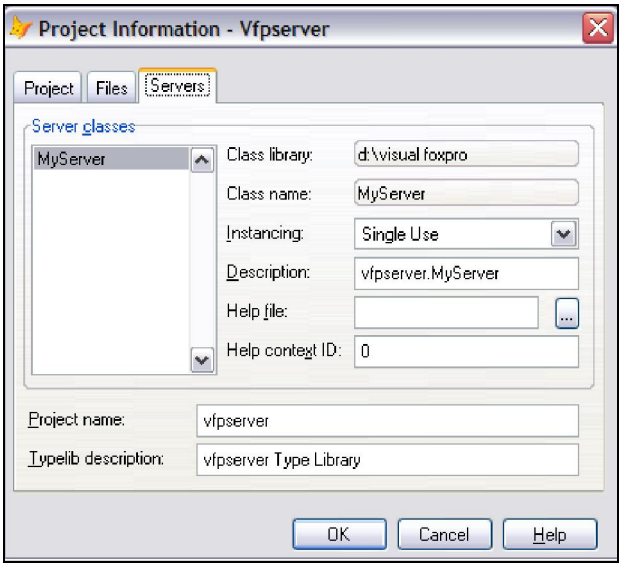

Рис. 16.8. Диалоговое окно **Project Information**

По умолчанию для EXE COM-сервера предлагается вариант **Single Use**. Это вполне оправданный выбор, т. к. EXE COM-сервер выполняется в собственном процессе, и для каждого клиента будет создан отдельный процесс.

Нажмите кнопку **OK**. Созданный нами сервер готов к работе.

> ***Замечание***
>
> Если вы изменили предлагаемые по умолчанию значения некоторых полей вкладки, то заново скомпонуйте EXE-модуль.

Откройте папку, в которой находятся файлы проекта. Кроме модуля vfpserver.exe, там вы найдете файлы vfpserver.vbr и vfpserver.tlb.

Файл vfpserver.vbr содержит справочную информацию о ключах реестра Windows, которые были созданы при регистрации COM-объекта (листинг 16.2). Файл vfpserver.tlb — это библиотека типов, о которой мы поговорим чуть позже.

```
VB5SERVERINFO
VERSION=1.0.0
HKEY_CLASSES_ROOT\vfpserver.MyServer = vfpserver.MyServer
HKEY_CLASSES_ROOT\vfpserver.MyServer\NotInsertable
HKEY_CLASSES_ROOT\vfpserver.MyServer\CLSID = {D4BA1A94-8057-4891-BF65-
3668F8DAB938}
HKEY_CLASSES_ROOT\CLSID\{D4BA1A94-8057-4891-BF65-3668F8DAB938} =
vfpserver.MyServer
```

```
HKEY_CLASSES_ROOT\CLSID\{D4BA1A94-8057-4891-BF65-3668F8DAB938}\ProgId =
vfpserver.MyServer
HKEY_CLASSES_ROOT\CLSID\{D4BA1A94-8057-4891-BF65-3668F8DAB938}
\VersionIndependentProgId = vfpserver.MyServer
HKEY_CLASSES_ROOT\CLSID\{D4BA1A94-8057-4891-BF65-3668F8DAB938}
\LocalServer32 = vfpserver.exe /automation
HKEY_CLASSES_ROOT\CLSID\{D4BA1A94-8057-4891-BF65-3668F8DAB938}\TypeLib =
{9827E6A3-E9CE-45E4-9EC3-2F04ECEE70FA}
HKEY_CLASSES_ROOT\CLSID\{D4BA1A94-8057-4891-BF65-3668F8DAB938}\Version = 1.0
HKEY_CLASSES_ROOT\CLSID\{D4BA1A94-8057-4891-BF65-3668F8DAB938}\Foxruntime =
VFP9R.DLL
HKEY_CLASSES_ROOT\INTERFACE\{E2AAA111-072D-4960-B384-F68F6108946C} = MyServer
HKEY_CLASSES_ROOT\INTERFACE\{E2AAA111-072D-4960-B384-
F68F6108946C}\ProxyStubClsid = {00020424-0000-0000-C000-000000000046}
HKEY_CLASSES_ROOT\INTERFACE\{E2AAA111-072D-4960-B384-
F68F6108946C}\ProxyStubClsid32 = {00020424-0000-0000-C000-000000000046}
HKEY_CLASSES_ROOT\INTERFACE\{E2AAA111-072D-4960-B384-F68F6108946C}\TypeLib =
{9827E6A3-E9CE-45E4-9EC3-2F04ECEE70FA}
HKEY_CLASSES_ROOT\INTERFACE\{E2AAA111-072D-4960-B384-
F68F6108946C}\TypeLib\"Version" = 1.0
; TypeLibrary registration
HKEY_CLASSES_ROOT\TypeLib\{9827E6A3-E9CE-45E4-9EC3-2F04ECEE70FA}
HKEY_CLASSES_ROOT\TypeLib\{9827E6A3-E9CE-45E4-9EC3-2F04ECEE70FA}\1.0 = vfpserver
Type Library
HKEY_CLASSES_ROOT\TypeLib\{9827E6A3-E9CE-45E4-9EC3-2F04ECEE70FA}\1.0\0
\win32 = vfpserver.exe
HKEY_CLASSES_ROOT\TypeLib\{9827E6A3-E9CE-45E4-9EC3-2F04ECEE70FA}\1.0
\FLAGS = 0
```

Вы можете открыть реестр и убедиться, что все перечисленные в файле ключи созданы, а значения установлены (естественно, у вас будут другие значения глобальных идентификаторов).

Для просмотра библиотеки типов нам понадобится **Object Browser**.

Информация о классе в *Object Browser*

Для запуска компонента **Object Browser** выберите в главном меню Visual FoxPro пункт **Tools**, а в появившемся меню — пункт **Object Browser**.

Выполните настройку компонента, для чего нажмите на кнопку **Options**. В появившемся диалоговом окне **Object Browser Options** установите флажки так, как показано на рис. 16.9.

Теперь **Object Browser** будет показывать унаследованные объектом методы интерфейсов vTable и IDispath. Закройте окно.

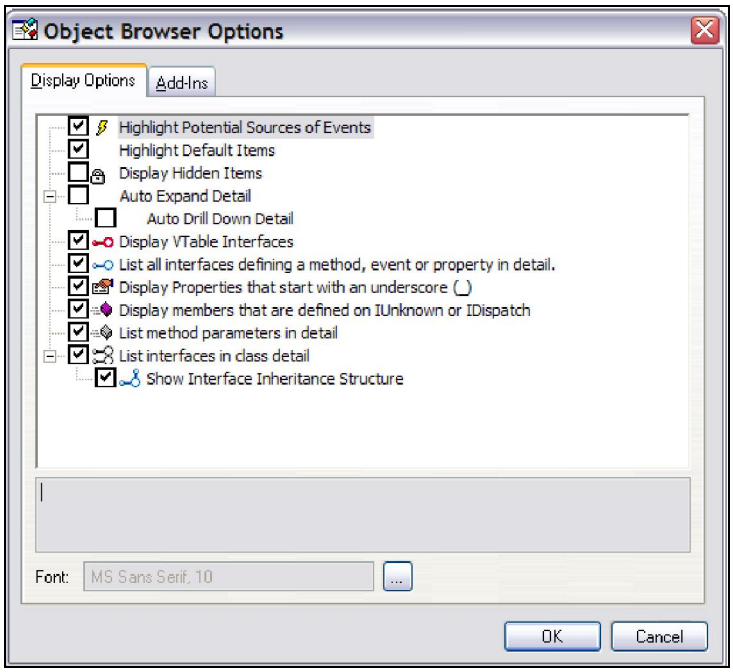

Рис. 16.9. Диалоговое окно настройки свойств компоненты **Object Browser**

В окне **Object Browser** нажмите на кнопку **Open Type Library** (она расположена справа вверху). В появившемся окне **Open** перейдите на вкладку **COM Libraries** (рис. 16.10).

Нажмите на кнопку **Browse…**, в диалоговом окне просмотра файлов установите в поле **Тип файлов** значение **Application**, перейдите в папку нашего проекта и выберите в ней файл vfpserver.exe. Информация из библиотеки типов загрузится в окно **Object Browser**.

Как видите, класс `vfpserver` является *сервером автоматизации* (он унаследован от интерфейса IDispath); помимо собственных методов класса `DoCmd` и `DoExec`, он включает в себя три метода, унаследованные от IUnknown (`AddRef`, `QueryInterface` и `Release`), а также методы, унаследованные от IDispath. Все эти методы *реализованы* в нашем классе, именно поэтому мы можем создавать объекты — экземпляры этого класса.

> ***Замечание***
>
> Точно такого же результата мы добились бы, загрузив в **Object Browser** файл библиотеки типов vfpserver.tlb. Загрузка модуля EXE (или DLL) в **Object Browser** возможна потому, что библиотека типов включена в него. Это так же упрощает процесс переноса COM-сервера на другие компьютеры, поскольку нет необходимости переносить и файл библиотеки типов.

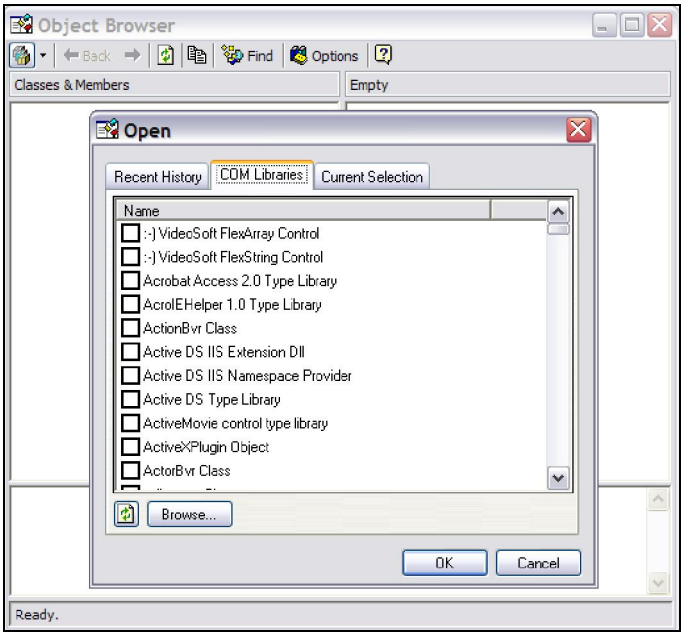

Рис. 16.10. Диалоговое окно **Open** компонента **Object Browser**

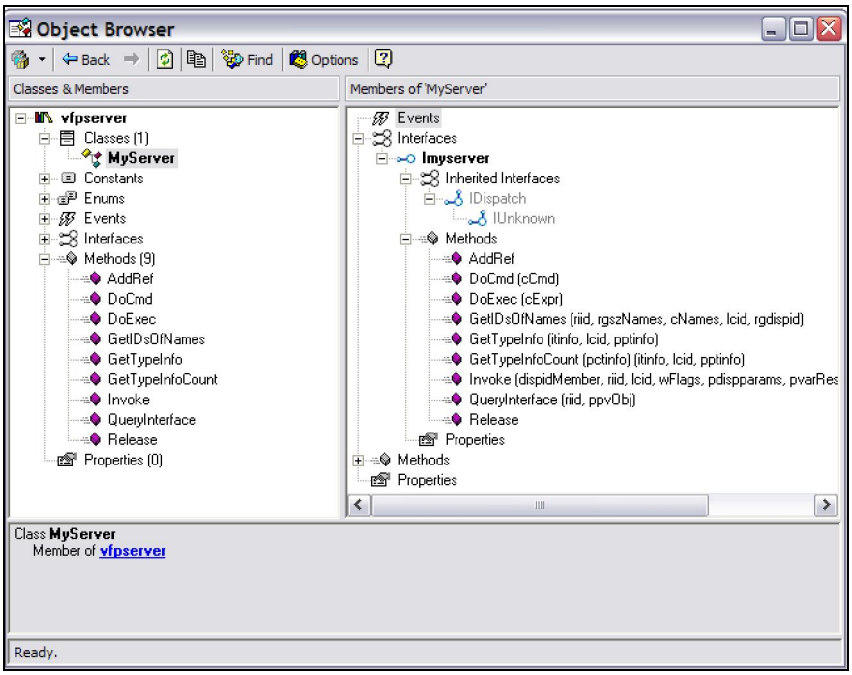

Рис. 16.11. Окно **Object Browser** с информацией о COM-сервере vfpserver

Тестирование COM-сервера vfpserver

При создании EXE-модуля COM-класса Visual FoxPro автоматически сгенерировал все необходимые идентификаторы и зарегистрировал класс в реестре Windows, поэтому мы сразу можем приступить к его тестированию. Но если вы решите установить COM-сервер на другом компьютере, то вы должны будете самостоятельно зарегистрировать его. Как это сделать, вы узнаете в конце главы.

А пока создайте новый проект, а в нем — программный файл, в который введите код из листинга 16.3.

```
oServer = CREATEOBJECT('vfpserver.myserver') && Создать объект
oServer.DoCMD('CREATE TABLE c:\test.dbf (num i, name c(50), ' + ;
              'amount b(1), cost b(2))')
oServer.DoCMD('INSERT INTO test VALUES (1, "Гвозди", 50, 120.5)')
oServer.DoCMD('INSERT INTO test VALUES (2, "Шурупы", 25, 152.12)')
oServer.DoCMD('INSERT INTO test VALUES (3, "Гайки", 1000, 22)')
oServer.DoCMD('CLOSE TABLES ALL')
oServer = null       && Уничтожить объект
```

Выполните этот код, а затем найдите на диске C: таблицу test.dbf, созданную COM-объектом. Откройте эту таблицу в окне **Browse** (рис. 16.12).

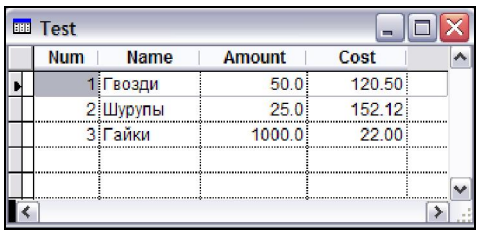

Рис. 16.12. Вид таблицы test.dbf в окне **Browse**

Как видите, метод `DoCMd` отработал правильно. Проверим работу метода `DoExec`.

```
oServer = CREATEOBJECT("vfpserver.myserver")
WITH oServer
   .DoCMD("USE c:\test.dbf IN 0")
   .DoCMD("PUBLIC nSumma")
   .DoCMD("nSumma = 0")
   .DoCMD("EXECSCRIPT('SELECT test' + CHR(13) + ;
```

```
        'SCAN' + CHR(13) + ;
        'nSumma = nSumma + test.amount * test.cost' + CHR(13) + ;
        'ENDSCAN')")
   = MESSAGEBOX(STR(.DoExec("nSumma"),10,2))
   .DoCMD("CLOSE TABLES ALL")
ENDWITH
oServer = null
```

В примере, показанном в листинге 16.4, возвращается сумма произведений полей **Amount** и **Cost** всех записей таблицы test.dbf. Обратите внимание на вызов на сервере встроенной функции `EXECSCRIPT` для выполнения кода, состоящего из нескольких команд, оформленных как одна строка.

Кстати, заметим, что переменная *`nSumma`*, объявленная как `PUBLIC` в методе сервера, существует *только внутри процесса сервера*. Клиент о ней ничего не знает.

Теперь попробуем прочитать данные из таблицы test.dbf в Microsoft Excel, написав макрос и вызвав из него созданный нами COM-сервер.

Для создания макроса запустите Microsoft Excel, в меню **Сервис** выберите пункт **Макрос** и в появившемся справа меню — пункт **Начать запись...** В окне **Запись макроса** (рис. 16.13) определите сочетание клавиш для вызова макроса, например, <Ctrl>+<A>.

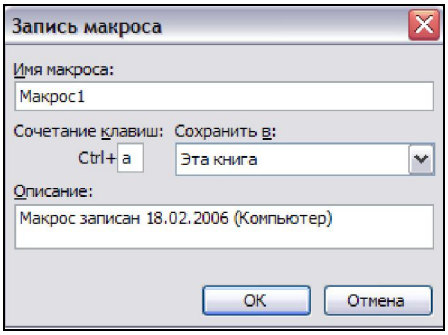

Рис. 16.13. Окно **Запись макроса** приложения Microsoft Excel

Вы сможете вызывать созданный макрос, просто нажав на выбранную комбинацию клавиш.

Нажмите на кнопку **OK**. В появившемся окошке — индикаторе записи сразу нажмите кнопку **Остановить запись**. Откройте макрос на редактирование, для чего в меню **Сервис** выберите пункт **Макрос**, в появившемся справа меню — пункт **Макросы...** и, в появившемся окне **Макрос**, выберите в списке созданный макрос и нажмите на кнопку **Изменить** (рис. 16.14).

Нажатием на эту кнопку вы запускаете встроенный в Microsoft Excel редактор VBA (Visual Basic For Applications). В окне редактора уже будет присутствовать сгенерированный код; измените его так, как показано в листинге 16.5.

```
Sub Макрос1()
' Макрос1 Макрос
' Макрос записан 18.02.2006 (Компьютер)
' Сочетание клавиш: Ctrl+A
'
Set ox = CreateObject("vfpserver.myserver")
ox.DoCmd ("USE D:\test.dbf")
n = ox.DoExec("reccount()")
nflds = ox.DoExec("fcount()")
For i = 1 To n
   For j = 1 To nflds
      cc = "evaluate(field(" & j & "))"
      Application.Sheets(1).Cells(i, j).Value = ox.DoExec(cc)
   Next
   ox.DoCmd ("skip")
Next
ox.DoCmd ("CLOSE TABLES ALL")
Set ox = Nothing
End Sub
```

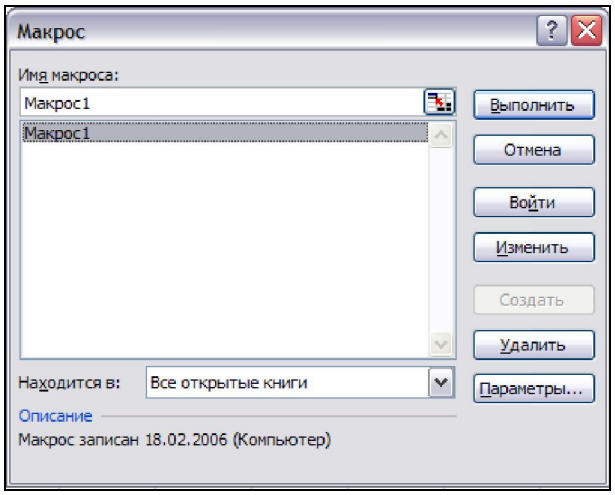

Рис. 16.14. Окно **Макрос**. Выбор макроса для редактирования

Вернитесь на лист книги Microsoft Excel и вызовите макрос, нажав на комбинацию клавиш (в нашем примере — <Ctrl>+<A>). Содержимое полей таблицы test.dbf будет занесено в ячейки листа книги Excel.

Особенности объявления COM-класса в Visual FoxPro

Рассмотренный выше пример COM-класса Visual FoxPro вполне работоспособен и может быть использован как образец для создания новых COM-классов. Он позволяет создавать объекты, способные выполнить практически любую команду Visual FoxPro. Но на самом деле мы рассмотрели далеко не все особенности, присущие COM-объектам. Ниже приведен синтаксис объявления COM-класса в Visual FoxPro:

```
DEFINE CLASS ClassName1 AS ParentClass OLEPUBLIC
   [[PROTECTED | HIDDEN] PropertyName1, PropertyName2 ...]
   [PEMName_COMATTRIB = Flags | DIMENSION PEMName_COMATTRIB[numElements]
   [PEMName_COMATTRIB[1] = Flags
       PEMName_COMATTRIB[2] = cHelpString
       PEMName_COMATTRIB[3] = cPropertyCapitalization
       PEMName_COMATTRIB[4] = cPropertyType
       PEMName_COMATTRIB[5] = nOptionalParams]]
   [[PROTECTED | HIDDEN] FUNCTION | PROCEDURE Name
      ([cParamName | cArrayName[] [AS Type] Type [@]]) [AS Type] Type
      [HELPSTRING cHelpString] ] [NODEFAULT]
         код_метода
   [ENDFUNC | ENDPROC]
ENDDEFINE
```

Как вы уже знаете, объявляет новый класс команда `DEFINE CLASS`. Для COM-класса вы должны указать в этой команде предложение `OLEBUBLIC`. Завершается объявление класса командой `ENDDEFINE`.

Свойства класса. Предложение *PEMName_COMATTRIB*

Сразу за объявлением класса должны следовать объявления его свойств. Как и для обычных классов, вы можете установить для каждого свойства область видимости, используя ключевые слова `PROTECTED` или `HIDDEN`. Если ни одно из этих ключевых слов не указано, то свойство объявляется как общедоступное (`PUBLIC`). Свойства, объявленные как `PROTECTED` или `HIDDEN`, не включаются в библиотеку типов и недоступны извне. Но более правильным будет определение видимости и способа доступа к свойству в предложении *PEMName*_COMATTRIB (*PEMName* — это имя свойства).

Предложение определяет атрибуты, записываемые в библиотеку типов для свойств или методов. Оно может быть издано в одном из следующих форматов:

```
PEMName_COMATTRIB = Flags
```

или

```
DIMENSION PEMName_COMATTRIB[NumElements]
```

В первом формате в предложении указывается флаг, значение которого вычисляется как сумма битовых масок, перечисленных в табл. 16.3.

Таблица 16.3. *Значения параметра* `Flags` *в предложении* `PEMName_COMATTRIB`

Константа в foxpro.h	Значение	Описание
COMATTRIB_RESTRICTED	0x00000001	Свойство не должно быть доступно из макроязыков. Этот флажок предназначен для функций уровня системы или функций, не отображающих окна просмотра. Средствами макроориентированного программирования (типа VBScript) не разрешается обращаться к этому элементу. Эти элементы обычно обрабатываются как _HIDDEN инструментальными средствами типа Visual Basic, основным отличием является невозможность связывания кода с этими элементами
COMATTRIB_HIDDEN	0x00000040	Свойство не должно быть показано пользователю, хотя оно существует и используется. COMATTRIB_HIDDEN означает, что свойство никогда нельзя показывать в окнах **Object Browser**, окнах просмотра свойств и т. д. Эта функция полезна для удаления единиц из объектной модели. Код может связываться с элементом, но пользователь никогда не будет знать, что элемент существует
COMATTRIB_NONBROWSABLE	0x00000400	Свойство появляется в окне **Object Browser**, но не появляется в окне просмотра свойств. COMATTRIB _NONBROWSABLE означает, что свойство не должно отобразиться в окне просмотра свойств. Это используется в обстоятельствах, в которых произошла бы ошибка, если свойство показывается в окне просмотра свойств. Раннее и позднее связывание налагает различные ограничения доступа. При раннем связывании клиент не будет способен читать значение свойства, доступного только для записи, или изменять значение свойства, доступного только для чтения, поскольку отсутствует вход в vtable. При позднем связывании клиент может все еще обращаться к propertyget для свойства, доступного только для записи, или propertyput для свойства, доступного только для чтения
COMATTRIB_READONLY	0x00100000	Свойство доступно только для чтения. Одновременное использование COMATTRIB_READONLY и COMATTRIB_WRITEONLY эквивалентно отсутствию доступа к свойству
COMATTRIB_WRITEONLY	0x00200000	Свойство доступно только для записи

В следующем фрагменте кода показано, как правильно использовать формат *PEMName*_COMATTRIB = *Flags*:

```
#INCLUDE foxpro.h
DEFINE CLASS myOLEClass AS Custom OLEPUBLIC
   MyProperty = "Test"
* Устанавливаем атрибут Flags для свойства MyProperty.
   myProperty_COMATTRIB = COMATTRIB_READONLY + COMATTRIB_WRITEONLY
ENDDEFINE
```

Второй формат предложения *PEMName*_COMATTRIB предполагает наличие массива, содержащего описание библиотечных атрибутов свойства PEMNAME. Массив может содержать до пяти элементов. Их назначение перечислено в табл. 16.4.

Таблица 16.4. *Описание элементов массива для PEMName_COMATTRIB*

Элемент массива	Описание	Тип данных	Комментарий
1	Флаг атрибутов	Целое число	Значения флага приведены в табл. 16.3.
2	Строка справки	Строка	Строка, сохраняемая в библиотеке типов для описания свойства (для справки о методах используйте предложение HELPSTRING)
3	Регистр	Строка	Учитывает регистр символов в имени свойства. Если не используется, то Visual FoxPro записывает имя свойства в библиотеке типов в верхнем регистре
4	Тип свойства	Строка	Определяет тип данных свойства как строку символов, которая появится в библиотеке типов. Функционирует аналогично предложению AS TYPE для методов (см. ниже)
5	Количество параметров	Целое число	Используется только при описании метода. Определяет число необязательных параметров в методе. Например, если это значение равно 2 для метода с 5 параметрами, то последние 3 параметра — необязательные. При позднем связывании значения по умолчанию для необязательных параметров у клиента будут иметь значение "ложь" (.F.), и функция PCOUNT() точно отразит истинное число переданных параметров. При раннем связывании значения по умолчанию для необязательных параметров будут всегда пустой строкой, и PCOUNT() будет возвращать общее количество параметров для метода, а не переданное количество. Если вы определяете меньшее количество параметров, чем фактическое число передаваемых параметров, то параметры с номером, большим, чем объявлено, считаются необязательными

В следующем фрагменте кода показано использование массива свойств в предложении *PEMName*_COMATTRIB:

```
#INCLUDE foxpro.h
DEFINE CLASS myOLEClass AS Custom OLEPUBLIC
   MyProperty = 5.2
* Установка COM-атрибутов для свойства MyProperty.
   DIMENSION MyProperty_COMATTRIB[4]
   myProperty_COMATTRIB[1] = COMATTRIB_READONLY
   myProperty_COMATTRIB[2] = "Свойство доступно только для чтения"
   myProperty_COMATTRIB[3] = "MyProperty" && учитывать регистр для имени
   myProperty_COMATTRIB[4] = "Float"      && Тип данных свойства
ENDDEFINE
```

Методы класса

При создании COM-класса вы можете определить область видимости его методов, используя ключевые слова PROTECTED и HIDDEN. В описании метода вы можете указать тип возвращаемых данных и типы данных получаемых методом параметров.

> ***ЗАМЕЧАНИЕ***
>
> Объект, создаваемый из COM-класса, код которого показан в листинге 16.1, использует тип данных Variant как для передаваемых методу параметров, так и для типа данных, возвращаемых методом. Использование типа Variant является основополагающим в технологии COM. Тем не менее дальнейшее развитие этой технологии, например, COM+ или Web Services, требует явного указания типа данных. Поэтому если вы собираетесь создавать приложение COM+ или работать с объектом через протокол SOAP в Web Services, то необходимо явно указывать типы данных.

Впрочем, ничего страшного не случится, если вы будете явно указывать типы данных для любого COM-сервера.

В табл. 16.5 приведены типы данных, объявляемые в COM-классе, и их соответствие типам данных в библиотеке типов.

Таблица 16.5. *Типы данных COM-класса*

Определяемый в VFP тип данных	Соответствующий ему тип данных в библиотеке типов	Определяемый в VFP тип данных	Соответствующий ему тип данных в библиотеке типов
Array	SAFEARRAY(type)	Integer	LONG
BinaryMemo	VARIANT	Logical	VARIANT_BOOL
Boolean	VARIANT_BOOL	Long	LONG
Byte	Unsigned char	Memo	VARIANT
Character	BSTR	Number	DOUBLE

Currency	CURRENCY	Object	IDispath
Date	DATE	Short	LONG
DateTime	DATE	Single	Single
Decimal	wchar_t	String	BSTR
Double	DOUBLE	Variant	VARIANT
Float	VARIANT	Void	VOID

Тип `BSTR` — это строка символов формата Unicode с нулевым символом в конце. К счастью, вы не должны заботиться о таком преобразовании — Visual FoxPro автоматически выполняет необходимое конвертирование.

Ниже показано объявление метода объекта, возвращающего данные типа `Long`. Методу передаются два параметра, первый типа `String`, второй — типа `Array`:

```
FUNCTION MyMethod(tcText as String, tcArray[] as Array) as Long
```

Если метод не возвращает никаких данных, то определите возвращаемый им тип как `VOID`:

```
FUNCTION MyMethod(tcInpString as String, tcOutString as String @) as Void
```

В этом фрагменте кода метод `MyMethod` первый параметр получает по значению, а второй — по ссылке (в описании параметра указан символ "@"), и не возвращает никаких значений.

При описании метода может быть использовано предложение `HELPSTRING`. Строка, указанная в этом предложении, будет размещена в библиотеке типов в части, описывающей метод. Вы можете увидеть такие строки при просмотре библиотеки типов объекта в **Object Browser**.

```
PROCEDURE Convert(cText as String) as String ;
   HELPSTRING "Конвертирование строки"
* Код метода
ENDPROC
```

Предложение `NODEFAULT` в описании метода запрещает выполнение кода аналогичного метода базового класса — родителя.

Выбор базового класса для создания COM-класса

В Visual FoxPro новый класс может быть создан путем наследования какого-либо базового класса или его потомка. В принципе, COM-класс может наследоваться от любого базового класса, тем не менее, следует придерживаться некоторых соглашений.

Как правило, на Visual FoxPro вы создаете COM-классы, которые предназначаются для обработки данных, т. е. для работы с таблицами. Базовый класс `Session` разработан именно для этих целей. Во-первых, этот класс создает приватную сессию данных. Во-вторых, при создании многопотокового COM-сервера этот класс использует модель апартаментов, что позволяет каждому потоку работать в своей сессии данных

(подробнее об этом будет рассказано далее в этой главе), и, в-третьих, COM-класс — потомок не наследует свойства и методы класса `Session`.

Конечно, вы можете использовать любой класс в качестве родительского, но при программном создании классов правильным будет выбор класса `Session` или `Custom`.

Работа с массивами

При работе с COM-сервером вам, скорее всего, понадобится передавать ему данные в виде массивов. Кроме того, вы наверняка захотите, чтобы COM возвращал вашему приложению-клиенту выбранные из таблицы данные, например, в виде массива.

В обычных классах Visual FoxPro такая проблема не возникает — вы просто передаете методу массив по ссылке. При использовании технологии COM такой способ не работает. Решение проблемы заключается в использовании встроенной функции `COMARRAY`. Эта функция определяет, *каким образом* массив передается COM-объекту. Вот ее синтаксис:

```
CurrentSetting = COMARRAY(oObject, nNewValue)
```

где:

- *`oObject`* — ссылка на COM-объект;
- *`nNewValue`* — параметр, определяющий порядок работы с массивом, который может принимать одно из значений, перечисленных в табл. 16.6.

Таблица 16.6. *Значения параметра `nNewValue` функции `COMARRAY()`*

Значение `nNewValue`	Описание
0	Массив передается по значению. Нумерация элементов массива начинается с нуля
1	Массив передается по значению. Нумерация элементов массива начинается с единицы
10	Массив передается по ссылке. Нумерация элементов массива начинается с нуля
11	Массив передается по ссылке. Нумерация элементов массива начинается с единицы

Задание индекса первого элемента массива равным нулю обычно применяется, если COM-класс был создан в Visual Basic.

Вы можете внутри метода COM-объекта изменять размеры массива. Если это нежелательно, то при вызове функции добавьте к значению *`nNewValue 100`*:

```
= COMARRAY(COMobj, 111)
```

В этом случае массив будет передаваться по ссылке, и изменить его размеры будет невозможно.

Если параметр *nNewValue* опущен, то функция возвращает текущую установку для указанного COM-объекта.

В следующем фрагменте кода показано, как создать COM-класс, выполняющий выборку из таблицы и возвращающий значения ее полей в виде массива.

```
DEFINE CLASS test AS Session OLEPUBLIC
PROCEDURE GetData(tcTbl as String, aResult[] as Array) as VOID
   USE (tcTbl) IN 0 ALIAS mytable
   SELECT * FROM mytable INTO ARRAY aResult
   USE IN mytable
ENDPROC
ENDDEFINE
```

В метод `GetData` передается строка с именем таблицы, из которой нужно выбрать данные, и ссылка на массив, в который нужно поместить результат выборки. Оператор `SELECT` выбирает данные из таблицы и помещает их в массив.

Код для тестирования COM-объекта:

```
LOCAL loCOM, lcTable, laResult[1,1]
lcTable = GETFILE("DBF")
loCOM = CREATEOBJECT("myserver.test")
= COMARRAY(loCOM, 11)
loCOM.GetData(lcTable, @laResult)
```

В тестовом примере после создания объекта — экземпляра COM-класса вызывается функция `COMARRAY`, определяющая, что массив передается по ссылке и что его размеры могут быть изменены. Обратите внимание на то, что массив объявлен как двумерный. Это условие всегда необходимо выполнять. После вызова метода `GetData` массив `laResult` будет содержать выбранные из таблицы данные. Теперь вы можете создать курсор и вставить в него полученные данные командой `INSERT`.

Еще один вариант получения ссылки на массив заключается в использовании специального синтаксиса оператора `RETURN`, разрешенного для COM-объектов:

```
RETURN @Array
```

Возвращаемый массив должен быть либо свойством класса, либо объявлен как `PUBLIC`. Элементы массива должны содержать данные одного типа.

Обработка ошибок

Ваш COM-объект может выполняться на другом компьютере, поэтому он не должен выводить никаких диалоговых окон, в том числе и окон с сообщениями об ошибке, по той простой причине, что появление такого диалогового окна приведет к остановке выполнения кода и ожиданию реакции на это сообщение. Представьте себе реакцию системного администратора, когда он увидит такое сообщение на своем сервере!

Запретить вывод такого диалогового окна можно при помощи функции `SYS(2335)`:

```
= SYS(2335, 0)
```

Когда параметр функции равен нулю, подавляется вывод всех диалоговых окон.

> ***Замечание***
>
> Если вы создаете COM-сервер для работы in process (т. е. как DLL-модуль), то подавление вывода диалоговых окон устанавливается автоматически.

Другое требование к COM-объекту — он не должен завершаться по ошибке.

Функция COMRETURNERROR транслирует сообщение о произошедшей на сервере ошибке приложению-клиенту; приложение-клиент, получив это сообщение, выводит диалоговое окно с сообщением об ошибке, предоставляя пользователю решить, продолжать работу COM-объекта или завершить его.

Синтаксис функции:

```
COMRETURNERROR(cExceptionSource, cExceptionText)
```

Передаваемые функции параметры:

- *cExceptionSource* — используется для описания источника, вызвавшего исключение, кода ошибки и т. п.;
- *cExceptionText* — содержит дополнительную информацию, позволяющую определить характер ошибки.

При возникновении ошибки управление передается методу Error объекта; именно в этом методе вы должны вызывать функцию COMRETURNERROR.

Добавьте в класс myserver (см. листинг 16.1) метод Error, в котором будет вызываться функция COMRETURNERROR:

```
PROCEDURE Error
LPARAMETERS nError, cMethod, nLine
   COMRETURNERROR(cMethod + ", строка " + LTRIM(STR(nLine)), ;
                  "Ошибка в vfpserver.myserver")
ENDPROC
```

Заново постройте COM-сервер. Для тестирования используйте следующий код:

```
oServer = CREATEOBJECT("vfpserver.myserver")
oServer.DoCMD("Этот вызов вызовет исключение")
```

Запустите тест на выполнение. Вы получите сообщение, показанное на рис. 16.15.

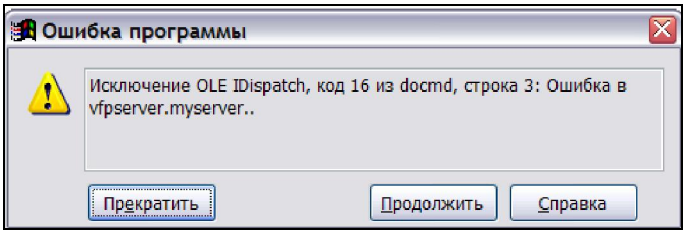

Рис. 16.15. Сообщение об ошибке COM-сервера, выводимое в приложении-клиенте

Если вы нажмете на кнопку **Прекратить**, то выполнение COM-объекта будет прекращено. Если вы нажмете на кнопку **Продолжить**, то выполнение COM-объекта будет продолжено, и вы без проблем сможете вызывать его методы.

> ***Замечание***
>
> При использовании функции `COMRETURNERROR ()` в секции `CATCH` в инструкции `TRY ... ENDTRY` дальнейшее выполнение кода (в том числе и в секции `FINALLY`) прекращается и управление передается приложению-клиенту.

Региональные настройки

Возможна ситуация, когда ваш COM-сервер выполняется на компьютере, для которого используемая по умолчанию региональная настройка отличается от русскоязычной (кириллицы). Это может привести к тому, что возвращаемые сервером символьные данные будут "нечитаемыми". Решить эту проблему можно, явно указав код кодовой страницы и региона.

Visual FoxPro предоставляет встроенные функции `SYS(2300)`, `SYS(3004)` и `SYS(3005)`, позволяющие указать, какие региональные настройки должны использоваться COM-объектом.

Удобнее всего эти настройки выполнять в методе `Init` класса:

```
PROCEDURE Init
   SET DATE GERMAN
   SET CENTURY ON
   SET DELETED ON
   SET EXCLUSIVE OFF
   = SYS(2335, 0)          && Запретить вывод диалоговых окон
   IF CPCURRENT() <> 1251  && Если текущая кодовая страница — не 1251,
      = SYS(2300, 1251, 1) && то установить "правильную" страницу
   ENDIF
   IF SYS(3004) # '1049'   && Если регион — не Россия,
      = SYS(3005, 1049)    && то устанавливаем "правильный" регион
   ENDIF
ENDPROC
```

Помимо региональных установок, в методе `Init` нужно выполнять все необходимые настройки среды, как это и показано в коде.

Создание COM-класса в Конструкторе классов

Конструктор классов целесообразно использовать для создания COM-компонентов, поддерживающих графический интерфейс пользователя, таких как формы. Поэтому мы создадим с его помощью COM-сервер, фактически являющийся полноценным COM-приложением, выполняющим функции текстового редактора. Главное окно нашего сервера будет иметь строку меню и панель инструментов. Как и в обычном (не COM) приложении, вы сможете, используя опции меню и кнопки панели инструментов, вызывать различные диалоговые формы, являющиеся компонентами приложения. Но, в отличие от обычного приложения, вы сможете запускать наше COM-приложение *только* из других приложений-клиентов.

Итак, приступим.

Создайте в отдельной папке новый проект с именем **VfpTextEdit**. В Менеджере проектов выберите узел **Class Libraries** и нажмите на кнопку **New**. В диалоговом окне **New Class** дайте создаваемому классу имя `MyApplication`, а в качестве базового класса-родителя выберите `Form`.

После загрузки Конструктора классов выберите в главном меню Visual FoxPro пункт **Class**, а в выпавшем меню — пункт **Class Info**. Будет выведено диалоговое окно **Class Info** (рис. 16.16). Установите в этом окне флажок **OLE Public**. Установкой этого флажка вы указываете Конструктору классов, что будет создаваться COM-класс.

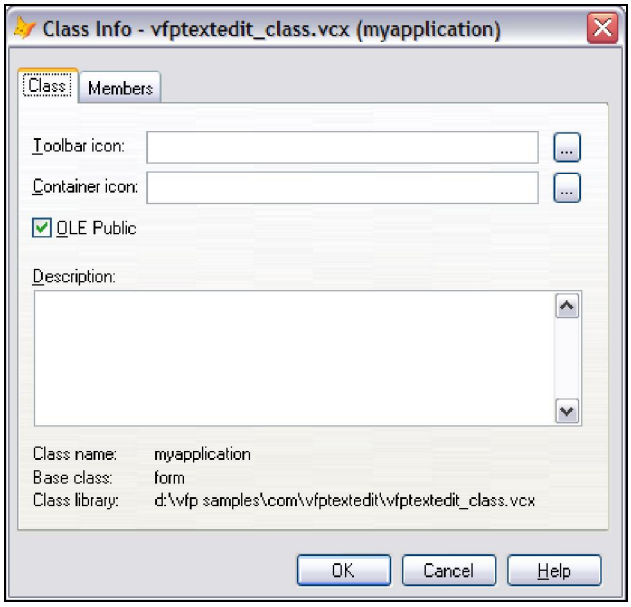

Рис. 16.16. Определение создаваемого класса как COM-класса

Разместите на форме управляющий элемент **EditBox** (Конструктор классов присвоит ему имя *Edit1*) и присвойте его свойству `Anchor` значение 15. Измените размеры элемента так, чтобы он занял всю клиентскую область окна.

В отличие от обычного приложения Visual FoxPro, которое может не иметь собственного главного окна и выполняться в главном окне Visual FoxPro, визуальное COM-приложение всегда выполняется в собственном главном окне, не имеющем родительского окна — т. е. в форме верхнего уровня. Поэтому присвойте свойству `ShowWindow` формы значение 2 (As Top Level Form), свойству `Caption` — значение "VFP редактор текстов", а в методе `Init` введите код создания меню (листинг 16.6).

```
PUBLIC goMainForm
goMainForm = this
DEFINE MENU MainMenu IN (this.name) BAR
DEFINE PAD mnu_File OF MainMenu PROMPT "Файл"
ON PAD mnu_File OF MainMenu ACTIVATE POPUP pad_File
DEFINE POPUP pad_File MARGIN RELATIVE SHADOW
DEFINE BAR 1 OF pad_File PROMPT "Открыть"
DEFINE BAR 2 OF pad_File PROMPT "Сохранить"
DEFINE BAR 3 OF pad_File PROMPT "\-"
DEFINE BAR 4 OF pad_File PROMPT "Шрифт"
DEFINE BAR 5 OF pad_File PROMPT "\-"
```

```
DEFINE BAR 6 OF pad_File PROMPT "Выход"
ON SELECTION BAR 1 OF pad_File goMainForm.Operation(1)
ON SELECTION BAR 2 OF pad_File goMainForm.Operation(2)
ON SELECTION BAR 4 OF pad_File goMainForm.Operation(3)
ON SELECTION BAR 6 OF pad_File goMainForm.Release()
ACTIVATE MENU MainMenu NOWAIT
```

Обратите внимание на объявление в методе глобальной переменной *goMainForm*. Эта переменная будет использоваться в нашем COM-приложении как ссылка на главную форму, т. е. вы сможете обратиться к любому свойству и методы формы, используя следующий синтаксис:

```
goMainForm.Method(parameters...)
```

При выборе первых трех пунктов меню будет вызываться метод `Operations`, который необходимо добавить в форму. Его код показан в листинге 16.7.

```
LPARAMETERS tnOperation
LOCAL lcFile
DO CASE
   CASE tnOperation = 1
      lcFile = GETFILE("TXT")
      IF !EMPTY(lcFile)
          this.Edit1.Value = FILETOSTR(lcFile)
      ENDIF
   CASE tnOperation = 2
      lcFile = PUTFILE("Сохранить в", "", "TXT")
      IF !EMPTY(lcFile)
          STRTOFILE(this.Edit1.Value, lcFile)
      ENDIF
   CASE tnOperation = 3
* Здесь позже будет вставлен код вызова формы для определения шрифта.
ENDCASE
```

При выборе пункта меню **Выход** будет вызван метод `Release` формы, что приведет к завершению COM-приложения.

Закройте Конструктор классов. В Менеджере проектов нажмите на кнопку **Build**, в появившемся диалоговом окне **Build Options** выберите **Win32 executable/COM server (exe)**. Сохраните COM-сервер в файле vfptextedit.exe.

Тестирование: клиент на Visual FoxPro

В листинге 16.8 приведен код, создающий форму, которая используется для тестирования COM-сервера.

```
PUBLIC goForm
goForm = CREATEOBJECT("MyTestForm")
goForm.Show()

DEFINE CLASS MyTestForm AS form   && Класс формы для тестирования
   Top = 0
   Left = 0
   Height = 120
   Width = 535
   Caption = "Тест"
   oApp = .F.
   Name = "Form1"
   ADD OBJECT command1 AS commandbutton WITH ;
      Top = 2, Left = 10, Height = 25, Width = 170, ;
      Caption = "Создать COM-объект",  Name = "Command1"
   ADD OBJECT command2 AS commandbutton WITH ;
      Top = 32, Left = 10, Height = 25, Width = 170, ;
      Caption = "Открыть файл", Name = "Command2"
   ADD OBJECT command3 AS commandbutton WITH ;
      Top = 62, Left = 10, Height = 25, Width = 170, ;
      Caption = "Сохранить файл", Name = "Command3"
   ADD OBJECT command4 AS commandbutton WITH ;
      Top = 92, Left = 10, Height = 25, Width = 170, ;
      Caption = "Уничтожить COM-объект", Name = "Command4"
   ADD OBJECT edit1 AS editbox WITH ;
      Height = 85, Left = 183, Top = 2, Width = 352, Name = "Edit1"
   ADD OBJECT command5 AS commandbutton WITH ;
      Top = 92, Left = 188, Height = 25, Width = 170, ;
      Caption = "Отправить текст на сервер", Name = "Command5"
   ADD OBJECT command6 AS commandbutton WITH ;
      Top = 92, Left = 359, Height = 25, Width = 170, ;
      Caption = "Получить текст с сервера", Name = "Command6"
* Создание объекта — экземпляра COM-класса
* по его программному идентификатору (ProgID)
   PROCEDURE command1.Click
      thisform.oApp = CREATEOBJECT("vfptextedit.myapplication")
      thisform.oApp.visible = .t.
   ENDPROC
   PROCEDURE command2.Click
      thisform.oApp.Operation(1) && Вызов метода Operation COM-объекта
   ENDPROC
   PROCEDURE command3.Click
      thisform.oApp.Operation(2) && Вызов метода Operation COM-объекта
   ENDPROC
   PROCEDURE command4.Click
```

```
      thisform.oApp = .f.        && Уничтожение COM-объекта
   ENDPROC
   PROCEDURE command5.Click
      thisform.oApp.Edit1.value = thisform.Edit1.Value
   ENDPROC
   PROCEDURE command6.Click
      thisform.Edit1.Value = thisform.oApp.Edit1.value
   ENDPROC
ENDDEFINE
```

Сохраните этот код в процедурном файле и запустите его на выполнение. В появившейся форме (рис. 16.17) нажмите на кнопку **Создать COM объект**.

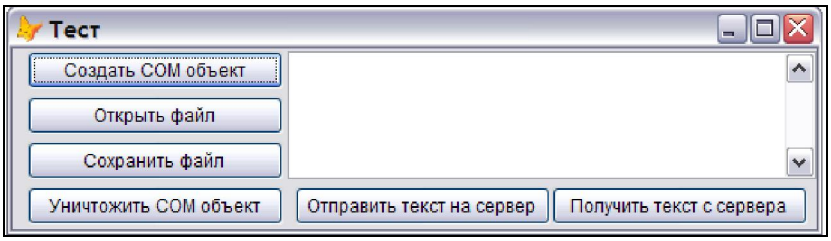

Рис. 16.17. Тестовая форма для проверки визуального COM-сервера

На экране появится окно нашего COM-приложения; его заголовок — "VFP редактор текстов". Разместите окна Visual FoxPro и COM-приложения так, чтобы они не перекрывали (или только частично перекрывали) друг друга (как, например, на рис. 16.18) — это нужно для того, чтобы формы "не сворачивались".

Наберите любой текст в поле `EditBox` тестовой формы и нажмите на кнопку **Отправить текст на сервер**. Набранный текст отобразится в окне сервера. Протестируйте сервер, чтобы убедиться, что все команды выполняются правильно.

Вы можете перейти в окно COM-приложения и работать в нем, как в обычном приложении. Завершите выполнение COM-сервера, нажав на кнопку **Close** в заголовке его главного окна, или выбрав пункт **Выход** в меню **Файл**, либо нажав на кнопку **Уничтожить COM объект** в тестовой форме.

Повторите эксперимент, предварительно загрузив Диспетчер задач Windows (для этого нужно одновременно нажать на клавиши <Ctrl>+<Shift>+<Esc>). В окне Диспетчера задач перейдите на вкладку **Процессы**. Убедитесь, что после запуска COM-приложения из тестовой формы строка **vfptextedit.exe** появляется в списке процессов, а при уничтожении — исчезает.

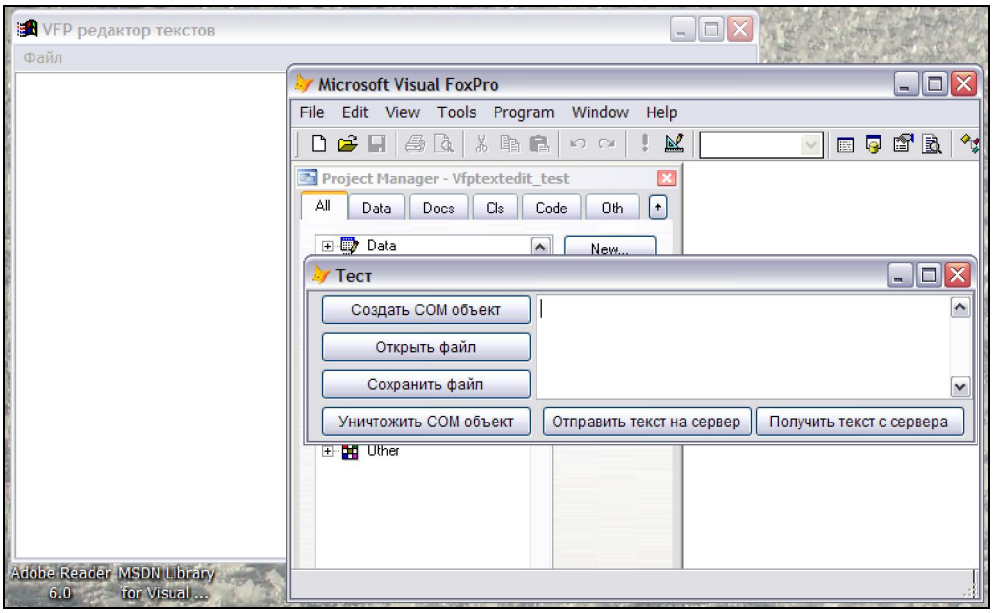

Рис. 16.18. Подготовка к тестированию COM-приложения из Visual FoxPro

Тестирование: клиент — Microsoft Excel

Запустите Microsoft Excel и создайте в нем три пустых макроса; первый макрос свяжите с комбинацией клавиш <Ctrl>+<A>, второй — с комбинацией <Ctrl>+<B> и третий — с комбинацией <Ctrl>+<E>. Перейдите в редактор кода VBA и измените код макросов так, как показано в листинге 16.9.

```
Public goApp
Sub Макрос1()
' Макрос1 Макрос
' Сочетание клавиш: Ctrl+a
'
   Set goApp = CreateObject("vfptextedit.myapplication")
   goApp.Visible = True
End Sub
Sub Макрос2()
' Макрос2 Макрос
' Сочетание клавиш: Ctrl+b
'
   goApp.Operation (1)
End Sub
```

```
Sub Макрос3()
' Макрос3 Макрос
' Сочетание клавиш: Ctrl+e
'
   goApp.Release
End Sub
```

Имейте в виду, что глобальная переменная `goApp` должна быть объявлена в секции `Declarations` окна редактора кода VBA.

Процедура `Макрос1()` создает объект — экземпляр COM-класса. Процедура `Макрос2()` вызывает на сервере диалоговое окно **Open**. `Макрос3()` выгружает COM-сервер.

Вернитесь из редактора кода VBA к книге Microsoft Excel и нажмите на комбинацию клавиш <Ctrl>+<A>. Сервер myapplication загрузится, и вы увидите на экране его окно. Как и при тестировании из Visual FoxPro, разнесите окна Microsoft Excel и сервера так, чтобы они не перекрывались или перекрывались частично. Для выполнения макросов необходимо, чтобы окно Microsoft Excel было активным.

Окончательная доработка приложения *VfpTextEdit*

Как правило, приложения, поддерживающие графический интерфейс пользователя, не ограничиваются одной главной формой. COM-приложение, как и любое другое приложение, должно предоставлять пользователю возможность вызывать различные формы, реализующие его функциональность. Все, что для этого требуется, — это добавить в ту же самую библиотеку классов, в которой находится наш COM-класс, другие необходимые классы, и создавать объекты — экземпляры этих классов по ходу выполнения приложения. При этом все эти дополнительные классы должны быть обычными, а не COM-классами; дело в том, что они образуют иерархию классов приложения, на вершине которой находится единственный COM-класс — форма верхнего уровня, реализующая главное окно COM-приложения.

Когда вы компонуете модуль COM-класса из библиотеки классов, то Visual FoxPro включает эту библиотеку в состав исполняемого модуля, следовательно, все классы, включенные в библиотеку, доступны из любого места программы, поэтому для создания новых объектов-контейнеров типа форм можно использовать функцию `CREATEOBJECT()`, а для управляющих элементов — метод `AddObject` объекта-контейнера. Особое внимание нужно обратить на то, что для всех включаемых в проект форм свойство `ShowWindow` должно быть равно единице, т. е. каждая дочерняя форма выполняется в форме верхнего уровня (действительно, при работе COM-приложения главное окно Visual FoxPro просто не создается). Это относится и к панелям инструментов. Если вы забудете установить это свойство, то такая форма просто не сможет себя "нарисовать", но при этом, если она объявлена как модальная, "подвесит" ваше COM-приложение.

Если вы внимательно изучили материал *главы 11*, то вам не составит труда добавить в библиотеку классов панель инструментов и диалоговую немодальную форму, поз-

воляющую устанавливать параметры шрифта. Для управления свойствами расположенного на главной форме управляющего элемента `EditBox` используйте глобальную переменную `goMainForm`, определенную в методе `Init` главной формы. На рис. 16.19 показано, как в нашем примере выглядит главное окно COM-приложения с вызванной формой установки параметров шрифта, а в листинге 16.10 показан код метода `Click` кнопки **Установить**.

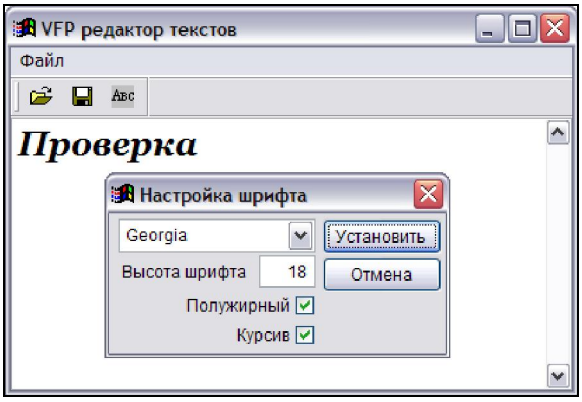

Рис. 16.19. Настройка параметров шрифта в COM-приложении

```
WITH goMainForm.Edit1
   .FontName = thisform.ComboFontName.Value
   .FontSize = thisform.TextFontSize.Value
   .FontItalic = thisform.CheckItalic.value
   .FontBold =  thisform.CheckBold.value
ENDWITH
```

Существует еще один момент, на который нужно обратить внимание, — это определение папки, которая будет использоваться COM-сервером по умолчанию. В листинге 16.11 показан код, который необходимо включить в метод `Load` главной формы для того, чтобы COM-приложение использовало по умолчанию ту папку, из которой загружается EXE-модуль сервера.

```
LOCAL lcPath
lcPath = SYS(16)
lcPath = JUSTPATH(SUBSTR(lcPath, AT(":",lcPath)-1))
SET DEFAULT TO (lcPath)
```

Для чего нужно определять папку, используемую по умолчанию? Ну, например, для того, чтобы ваше COM-приложение могло печатать отчеты. Поместите файлы отчетов (FRX и FRT) в эту папку, и приложение "увидит" их в процессе выполнения.

Сокрытие "лишних" методов и свойств

Описания всех открытых свойств и методов COM-класса включаются в библиотеку типов и будут доступны приложению-клиенту, создавшему COM-объект. Класс формы, который лежит в основании созданного нами приложения, как и любой другой визуальный класс, экспонирует большой набор свойств и методов, доступ к которым из приложения-клиента не только не нужен, то иногда даже опасен.

Поэтому необходимо изменить уровень видимости для большинства свойств и методов формы на `PROTECTED`. Необходимо оставить открытыми только те методы и свойства, которые по замыслу должны быть доступны из клиента или из дочерних форм, вызываемых в процессе выполнения COM-приложения. Так же не рекомендуется "закрывать" свойство `HWND` окна.

Для изменения уровня видимости свойств и методов загрузите главную форму нашего приложения в Конструктор классов, после чего в меню **Class** выберите пункт **Edit Property/Method...**. Диалоговое окно **Edit Property/Method** показано на рис. 16.20.

Просмотрите каждое свойство/метод и при необходимости установите для него в поле `Visibility` значение `Protected`. После выполнения этой операции заново скомпонуйте исполняемый модуль COM-сервера.

Все коды проекта VfpTextEdit вы найдете на прилагаемом к книге компакт-диске.

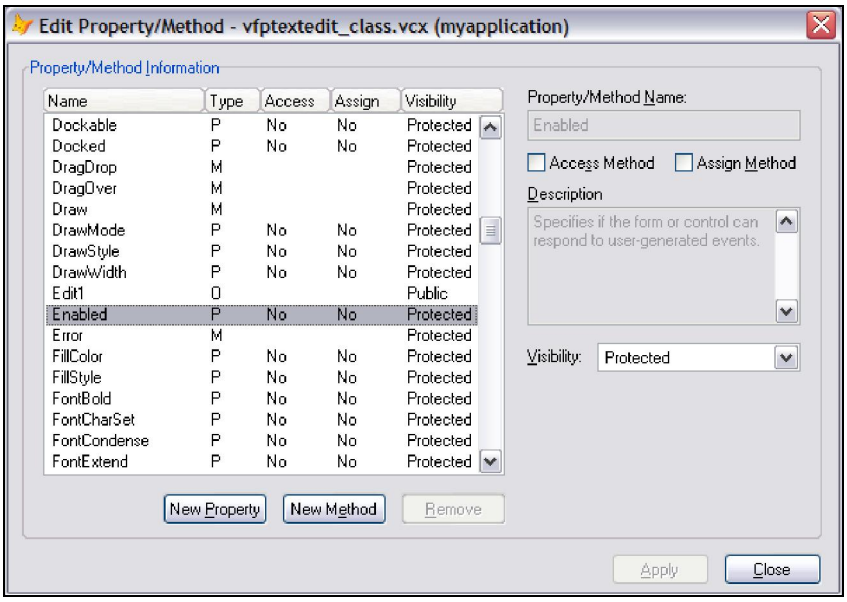

Рис. 16.20. Окно **Edit Property/Method** для класса `myapplication`

Распределенные приложения

Согласитесь, достаточно сложно понять необходимость создания на Visual FoxPro COM-классов, которые будут работать на том же компьютере, на котором выполняется приложение пользователя — ну если только эти COM не разрабатываются для использования другими (не Visual FoxPro) приложениями.

Другое дело, если вы создаете распределенные приложения, т. е. приложения, состоящие из нескольких компонентов, выполняющихся на разных компьютерах вашей локальной сети. Ваше клиентское приложение создает объект — экземпляр COM-сервера на другом компьютере, например, на том, где находятся базы данных, и начинает работать с ним, как, например, с сервером баз данных типа Microsoft SQL Server. Таким образом, вполне доступными средствами реализуется технология "клиент-сервер".

Для создания объекта на любом компьютере локальной сети используется функция `CREATEOBJECTEX`. Вы уже встречались с ней при обсуждении вопросов раннего и позднего связывания. Сейчас мы будем использовать эту функцию по ее основному назначению:

```
oServer = CREATEOBJECTEX(ClsID, ServerName [, IID)
```

Передаваемые функции параметры:

- *ClsID* — это ClsID COM-класса; при создании COM-объекта на другом компьютере для значения этого параметра нельзя использовать ProgID, т. к. он не обеспечивает уникальность компонента;
- *ServerName* — имя компьютера в вашей локальной сети в формате UNC (Universal Naming Convention), например: "//ServerDB";
- *IID* — необязательный параметр, позволяет передать функции IID интерфейса.

Для того чтобы объект был создан успешно, необходимо, чтобы COM-класс был определенным образом зарегистрирован на компьютере-сервере и вы имели необходимые права доступа к нему. Настройка прав доступа и ряд других установок зависят от используемой модели распределенного приложения: DCOM или COM+.

Распределенные COM (DCOM)

Такая аббревиатура (DCOM, Distributed Component Object Model) применяется к COM-серверам, выполняющимся на другом (не вашем) компьютере сети. Сервер DCOM всегда выполняется в собственном процессе (out-of-process) и поэтому создается как EXE-модуль.

Перед тем как приступить к настройке DCOM, убедитесь, что ваш COM-сервер зарегистрирован в системе. Регистрация COM EXE-модуля выполняется командой

```
имя_файла_компонента /regserver
```

которую вы должны выдать в командной строке Windows, например:

```
myserver.exe /regserver
```

Для настройки DCOM щелкните правой кнопкой мыши на значке **Мой компьютер** на рабочем столе Windows и в появившемся меню выберите пункт **Управление**. В появившемся диалоговом окне **Управление компьютером** откройте оснастку **Службы компонентов** и в ней выберите узел **Настройка DCOM** (рис. 16.21).

> ***Замечание***
> Оснастка **Службы компонентов** по умолчанию доступна в Windows 2003 Server. Если вы хотите зарегистрировать DCOM в Windows XP, то вы должны самостоятельно подключить эту оснастку.

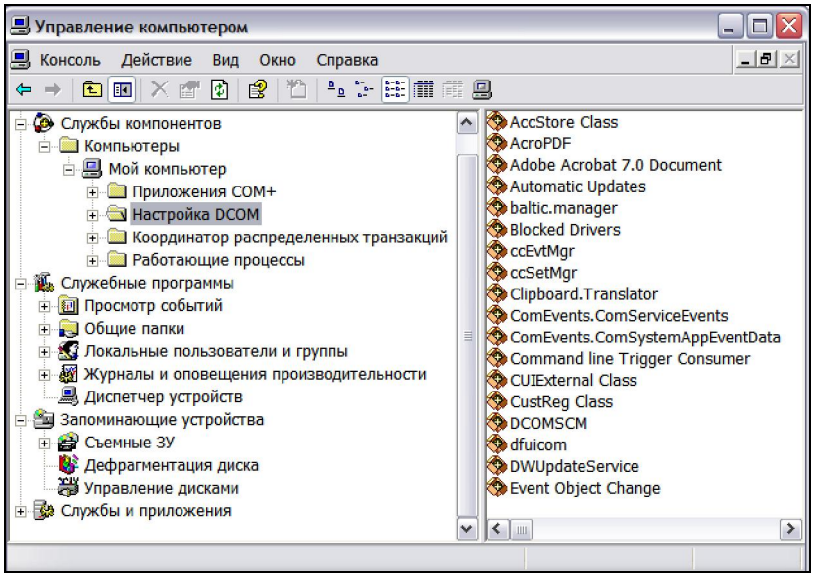

Рис. 16.21. Диалоговое окно **Управление компьютером**

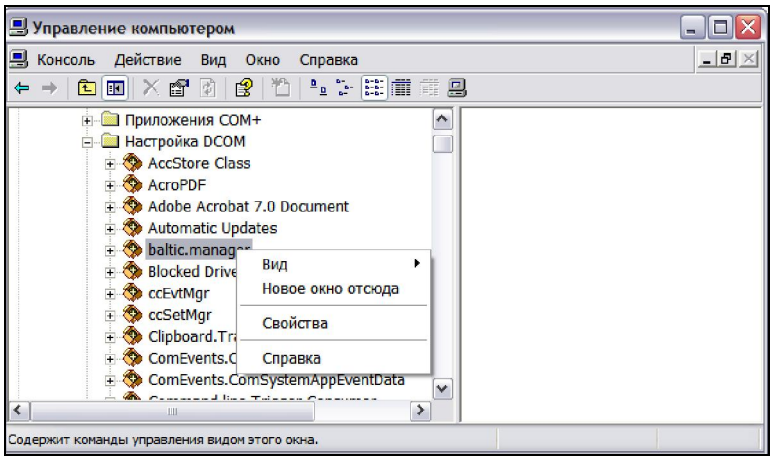

Рис. 16.22. Выбор объекта для настройки его свойств

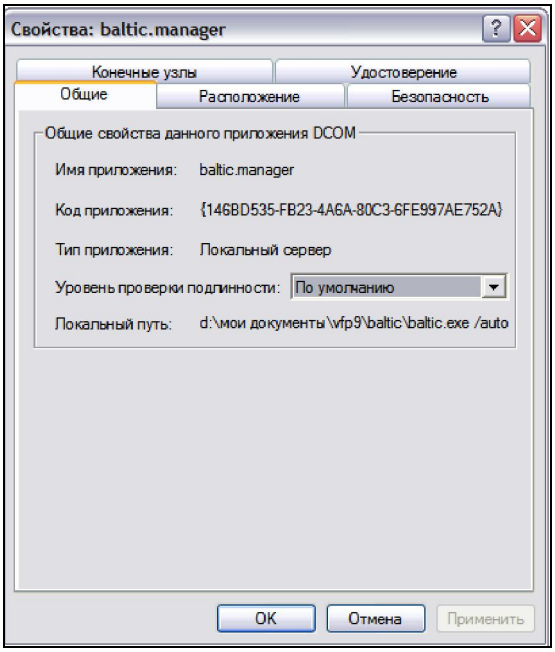

Рис. 16.23. Диалоговое окно настройки свойств DCOM в Windows XP

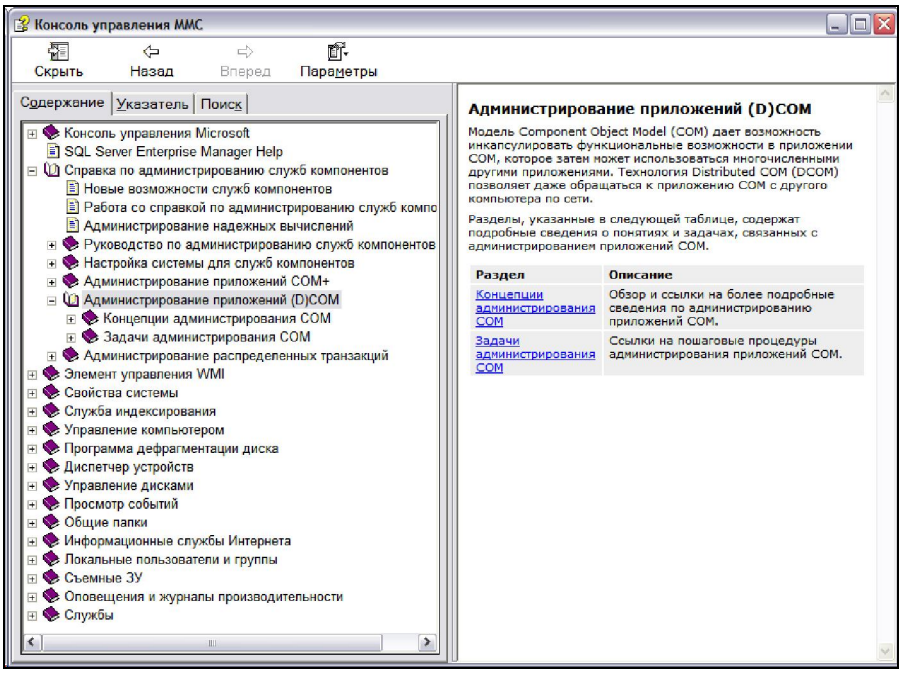

Рис. 16.24. Вид окна со справочной информацией

Откройте узел **Настройка DCOM** и в списке COM-объектов найдите ProgID вашего сервера. Щелкните по нему правой кнопкой мыши и в появившемся контекстном меню выберите пункт **Свойства** (рис. 16.22).

На экране появится диалоговое окно настройки свойств DCOM (рис. 16.23).

В этом окне выполните необходимые действия по настройке компонента.

Методика настройки несколько отличается в различных версиях Windows. Для получения подробной информации обратитесь к справочной документации, для чего в контекстном меню (рис. 16.24) выберите пункт **Справка**.

Приложения COM+

Технология DCOM имеет один существенный недостаток: для каждого клиента создается свой экземпляр COM-объекта, который существует до тех пор, пока клиент его не уничтожит. Это может привести к неоправданному увеличению требуемых ресурсов системы, когда объект необходим сразу большому количеству клиентов. Избежать этого позволяет технология COM+, в которой выполняется управление способом запуска и времени жизни объектов, причем множество клиентов используют один и тот же объект.

Visual FoxPro позволяет создавать такие многопотоковые (multi-threaded) серверы. Для создания такого компонента в диалоговом окне **Build Options** (см. рис. 16.7) выберите опцию **Multi-threaded COM Server (dll)**.

> ***Замечание***
>
> В принципе, необязательно создавать приложение COM+ для DLL-модуля. Но при этом вы должны учитывать, что такой компонент будет выполняться в процессе приложения-клиента и любой сбой внутри этого компонента приведет к неработоспособности самого приложения. Кроме того, технология COM+ при работе с потоками использует транзакции, что позволяет избежать множества проблем, связанных с одновременным доступом к данным.

Модели *StateLess* и *StateFull*

При использовании вашего COM в приложении COM+ компонент должен удовлетворять условиям, связанным с *сохранением среды окружения*. Дело в том, что в приложении COM+ компонент не находится в памяти компьютера постоянно; при обращении к методу такого компонента создается поток для выполнения этого метода, и если класс использует открытые (общедоступные) свойства, то при создании нового потока значения этих свойств теряются.

Модель **StateFull** предполагает наличие открытых свойств и общедоступных переменных, объявленных в классе; она по умолчанию применяется в DCOM, когда созданный объект в течение всего времени жизни находится в памяти, и значения

свойств и глобальных переменных сохраняются между вызовами методов этого объекта. То же можно сказать и о курсорах.

Модель **StateLess** предполагает, что объект не имеет никаких открытых свойств и общедоступных переменных. Именно эта модель должна использоваться при создании серверов как DLL-модулей.

Критические секции

При многопотоковости обращение к одним и тем же данным может одновременно выполняться из нескольких потоков. Метод синхронизации потоков, использующий критические секции, может применяться только для потоков внутри одного процесса.

Управление критическими секциями осуществляется посредством вызова функции `SYS(2336)`. Вот ее синтаксис:

```
SYS(2336 [, nAction])
```

Значения параметра *nAction* перечислены в табл. 16.7.

***Таблица 16.7.** Управление критическими секциями*

nAction	Описание
Нет	Возвращает текущее значение счетчика объявленных секций
1	Вызывает функцию Windows API `EnterCriticalSection`
2	Вызывает функцию Windows API `LeaveCriticalSection`
3	Вызывает функцию Windows API `LeaveCriticalSection` до тех пор, пока счетчик критических секций не станет равным нулю.

Функция возвращает символьное значение — байт, содержащий значение счетчика объявленных критических секций.

В листинге 16.12 приведен пример кода, реализующего критическую секцию для потока многопотокового in-process сервера:

```
SYS(2336,1)            && Начало кода критической секции
... код критической секции
SYS(2336,2)            && Завершение критической секции
```

Вызов функции `SYS(2336,1)` определяет начало критической секции. Вызываемая Windows API функция `EnterCriticalSection` проверяет, не выполняет ли уже какой-нибудь другой поток критическую секцию. Если нет, поток получает разрешение на выполнение своего критического кода. Если да, то поток, обратившийся с запросом,

переводится в состояние ожидания, пока поток, создавший критическую секцию, не освободит ее.

Вызов функции `SYS(2336,2)` из потока, владеющего критической секцией, сообщает операционной системе, что она может проверить, существует ли очередь на выполнение своей критической секции другими потоками; если да, то система выведет такой поток из состояния ожидания и предоставит ему процессорное время.

Апартаменты

Апартамент — это не процесс и не поток. Основное назначение апартаментов — разграничение доступа к данным из разных потоков. Смысл апартаментов заключается в том, что объекты — экземпляры одного и того же COM-класса, загруженные в разные потоки, не имеют доступа к данным друг друга.

Библиотека времени выполнения vfp9t.dll, разработанная специально для поддержки многопотоковых in-process серверов, реализует модель апартаментов, что позволяет устранить конфликты доступа к глобальным данным из различных потоков за счет создания копии таких данных для каждого апартамента, а использование базового класса `SESSION` для создания COM-объекта позволяет организовать отдельные сессии данных (private data session) внутри каждого потока.

В серверах, созданных на базе класса `Session` и скомпонованных как multi-threaded COM DLL, апартаменты поддерживаются автоматически.

Создание COM+-приложения

Процесс создания COM+-приложения достаточно трудоемок. Сначала вы должны создать "пустое" приложение COM+, а потом добавить в него необходимые компоненты.

Для создания новых пустых приложений COM+ использоваться программа администрирования служб компонентов. Исполнитель этой процедуры должен быть участником роли **Администратор** по отношению к системному приложению на конечном компьютере. В используемой по умолчанию конфигурации эта роль назначается только администраторам локальных компьютеров.

> ***Замечание***
>
> Серверное приложение COM+ можно установить только на компьютер, который работает под управлением Microsoft® Windows® 2000 или более поздних версий. Серверному приложению требуется платформа, поддерживающая службы компонентов.

Минимальной конфигурацией приложения считается конфигурация, не требующая обеспечения безопасности. При необходимости обеспечения безопасности приложения воспользуйтесь программой администрирования служб компонентов и установите безопасность на уровне процесса или добавьте пользователей к готовым ролям после создания приложения. Дополнительные сведения о настройке безопасности для приложения см. в разделе Администрирование безопасности приложения.

Замечание

Чтобы сетевые пользователи могли запускать приложение COM+ без аутентификации, в списки участников ролей приложения необходимо включить "Анонимного" пользователя. На сервере Windows Server 2003 по умолчанию "Анонимный" пользователь не входит в группу "Все".

Для создания нового приложения COM+ с помощью программы администрирования служб компонентов выполните следующие действия:

1. В дереве консоли программы администрирования служб компонентов в узле **Службы компонентов** (см. рис. 16.21) откройте узел **Приложения COM+** компьютера, на который требуется установить данное приложение.
2. Щелкните правой кнопкой мыши на папке **Приложения COM+** и выберите команду **Приложение** в меню **Создать**.
3. В окне **Мастера установки приложения COM+** нажмите кнопку **Далее**.
4. В диалоговом окне **Установка или создание нового приложения** выберите параметр **Создать пустое приложение**.

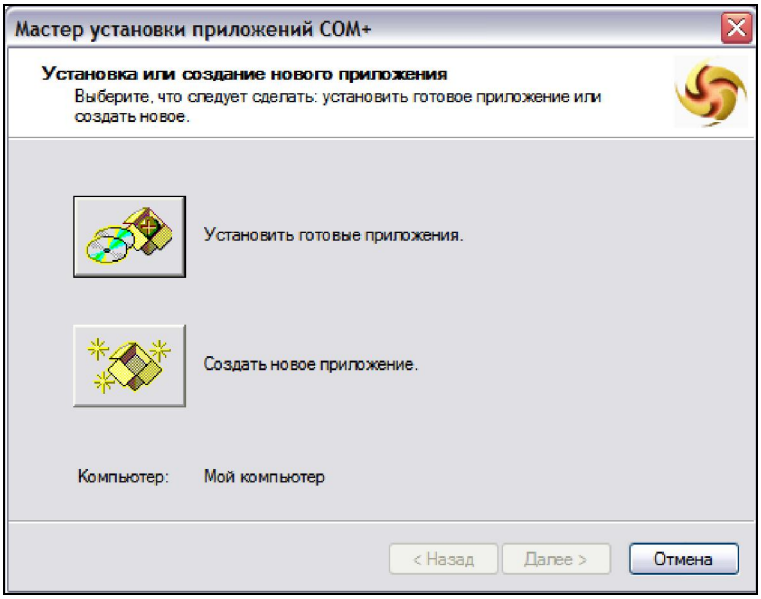

Рис. 16.25. Окно Мастера установки приложений COM+

5. В диалоговом окне **Создание пустого приложения** введите имя нового приложения и укажите, должны ли компоненты приложения при их вызове активизироваться в вызывающем процессе *(Библиотечное приложение)* или в отдельном серверном процессе *(Серверное приложение)*. Нажмите кнопку **Далее**.

6. В диалоговом окне **Удостоверение приложения** укажите удостоверение приложения и нажмите кнопку **Далее**.

> ***Замечание***
>
> По умолчанию используется запись **Взаимодействующий пользователь**. Взаимодействующий пользователь — это пользователь, вошедший в систему на исполняющем приложение компьютере с помощью учетной записи Windows. Можно выбрать другого пользователя. Для этого выберите параметр **Указанный пользователь** и введите сведения о конкретном пользователе или группе Windows.

7. В диалоговом окне **Добавление ролей приложения** укажите все роли, которые требуется создать для этого приложения. По умолчанию создается одна роль с именем **CreatorOwner**. Нажмите кнопку **Далее**.
8. В диалоговом окне **Назначение ролей пользователям** добавьте требуемых пользователей к созданным ролям. По умолчанию роль **CreatorOwner** назначается вашей учетной записи. Нажмите кнопку **Далее**.
9. Нажмите кнопку **Готово**.

Откройте узел **Приложения COM+** (см. рис. 16.21) и щелкните правой кнопкой мыши по имени созданного приложения. В появившемся контекстном меню выберите пункт **Свойства**. В диалоговом окне **Свойства** установите все необходимые параметры, а на вкладке **Активизация** в поле **Имя удаленного сервера** введите имя COM-сервера, а в поле **Корневая папка приложения** — полный путь к папке, в которой находится файл сервера.

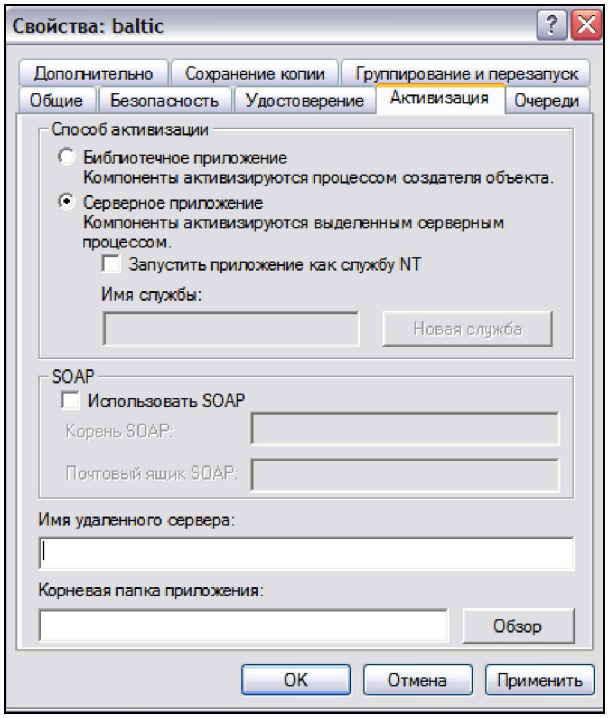

Рис. 16.26. Вкладка **Активизация** диалогового окна **Свойства**

Для получения более подробной информации по настройке COM+-приложений обратитесь к справочной документации.

Регистрация COM-серверов

Для того чтобы созданный вами COM-сервер мог успешно использоваться различными приложениями-клиентами, он должен быть зарегистрирован в реестре Windows.

Регистрация EXE COM-сервера выполняется командой

```
имя_файла_сервера /RegServer
```

Удаление сведений о таком сервере из Реестра выполняется командой

```
имя_файла_сервера /UnRegServer
```

Для регистрации DLL COM-серверов применяется специальная утилита regsvr32:

```
Regsvr32 имя_файла_сервера
```

Удаление информации о DLL COM-сервере из Реестра так же выполняет утилита regsvr32:

```
Regsvr32 /u имя_файла_сервера
```

Обратите внимание на то, что ключ `/u` указывается перед именем сервера.

Заключение

В этой главе вы получили как начальные теоретические, так и практические сведения об использовании технологии COM. В следующей главе мы продолжим знакомство с этой технологией на примере серверов автоматизации Microsoft Excel и Microsoft Word.

Конечно, невозможно осветить все аспекты применения COM в рамках одной главы; к сожалению, здесь не рассматривается технология Web Service и протокол SOAP, при использовании которого вы можете разворачивать свои COM-серверы на компьютерах в сети Интернет. Ничего не рассказано о способе взаимодействия клиента и сервера и способах управления потоками.

Но вы можете изучить эти аспекты самостоятельно, главное, чтобы после прочтения этой главы вы почувствовали все достоинства этой технологии и научились применять ее на практике!