ФЕДЕРАЛЬНОЕ АГЕНТСТВО ПО ОБРАЗОВАНИЮ
ГОСУДАРСТВЕННОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ
ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ
«ВОРОНЕЖСКИЙ ГОСУДАРСТВЕННЫЙ
УНИВЕРСИТЕТ»

Основы COM-технологий

Учебно-методическое пособие для вузов

Издательско-полиграфический центр
Воронежского государственного университета
2007

Утверждено научно-методическим советом факультета прикладной математики, информатики и механики 14 мая 2007, протокол № 9

Авторы: М.А. Артемов, А.А. Вахтин, Г.Э. Вощинская, В.Г. Рудалев

Рецензент зав. каф. ПиИТ ф-та ФКН ВГУ Н.А. Тюкачев

Учебное пособие подготовлено на кафедре программного обеспечения и администрирования информационных систем факультета прикладной математики, информатики и механики Воронежского государственного университета.

Для специальности 010503 – Математическое обеспечение и администрирование информационных систем.

Содержание

Введение

Пытаясь расширить возможности Windows, компания Microsoft столкнулась с необходимостью создания технологии, которая обеспечивала бы доступ приложений к объектам, расположенным вне приложения. Другими словами, приложения должны работать с внешними объектами с помощью методов, которые используются для работы с внутренними объектами. Кроме того, объекты не должны зависеть от языка программирования, к ним должен быть открыт доступ со стороны приложений не только локального, но и удаленного компьютера.

В качестве решения проблемы, связанной с доступом к объектам в пределах отдельного компьютера (приложение—приложение) и локальной сети (компьютер—компьютер), компания Microsoft разработала модели составных объектов, которые называются COM[1] (Component Object Model) и DCOM (Distributed COM).

В данном методическом пособии изложены основы этой технологии и продемонстрированы примеры создания и использования приложений, основанных на технологии COM.

I. Основные понятия и определения

1.1. Что такое COM. Основные свойства, отличающие COM от ООП

Технология объектно-ориентированного программирования основана на том, что приложение разбивается на объекты, которые взаимодействуют друг с другом внутри программы (рис. 1.1).

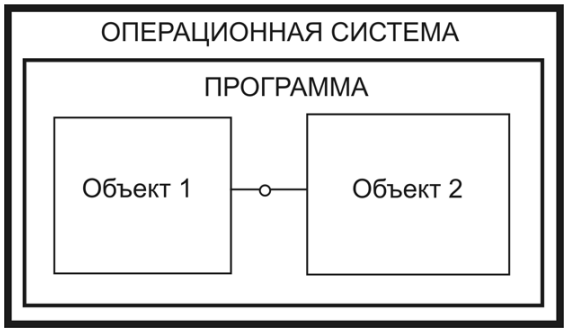

Рис. 1.1. Программа, сконструированная в ООП

[1] Начиная с Windows 2000 вводится новая технология, которая называется COM+. Это расширенные возможности технологии COM, поддерживающая архитектуру Windows DNA (Distributed interNet Application) [5].

Это дает возможность быстрой разработки и модификации программных продуктов с избежанием ошибок, так как программный код разбивается на отдельные блоки (объекты), связь между которыми осуществляется по установленным правилам (свойства и методы).

В COM-технологии объекты помещаются в отдельные исполняемые блоки (рис. 1.2), которыми могут являться динамически компонуемые библиотеки (DLL) или приложение (EXE).

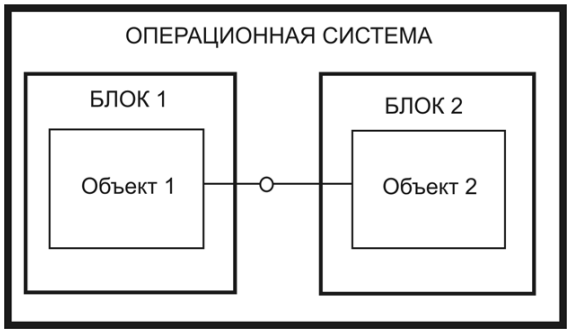

Рис. 1.2. Программа, сконструированная по архитектуре COM

Модель COM представляет собой инструкцию по созданию совместимых компонентов программного обеспечения и не является языком программирования, библиотекой программного кода или компилятором.

Технология COM основана на следующих основных концепциях.

1. *Уникальность и контекстная независимость компонент*. В системе не должно быть компонентов с одинаковым способом обращения и разными смысловыми назначениями.
2. *Инкапсуляция*. Реализация компонентов COM должна быть скрыта. Это необходимо для того, чтобы была независимость от языков программирования.
3. *Портативность и коммуникабельность*. Нужный компонент можно быстро найти динамически, где бы он ни находился.

Преимущество COM заключается в том, что это позволяет каждому разработчику сконцентрироваться на разработке своих компонент, которые могут обмениваться информацией независимо от языка программирования или инструментальных средств реализации. Более того, технология COM дает гибкие возможности динамической настройки и модификации программного продукта с учетом желаний пользователя и особенностей операционной системы.

1.2. Отношение «Клиент-сервер»

Пусть приложению требуется совершить какое-то действие или процесс, который ему неизвестен. Но есть компонент, находящийся в другом двоичном коде, который может это действие или процесс совершить. Если приложение *«знает»* о существовании и возможностях этого компонента, то оно *«заставляет»* компонент выполнить соответствующее действие.

Такое взаимодействие приложения и компонента называют отношением *«клиент-сервер»*, где приложение, играющее роль инициатора, называется *«клиентом»*, а исполняемый модуль, хранящий компонент, – *«сервером»*.

Существует два типа COM-серверов: *встроенные*, также их называют *внутренние*, и *внешние*. Встроенные COM-серверы это динамически компонуемые библиотеки (DLL), которые при загрузке в память компьютера отображаются на адресное пространство COM-клиента (программы). Внешние COM-серверы реализованы в виде исполняемых программ (exe-файлов), и, следовательно, они в процессе выполнения занимают в памяти компьютера адресное пространство, отдельное от клиентского приложения. Это приводит к некоторым нюансам при создании такого рода COM-серверов, о чем будет сказано в п. 2.8.

1.3. Уникальность компонент. GUID, CLSID

Чтобы приложение могло связаться с компонентом, необходимо знать имя объекта. Но тут возникает проблема совпадения имен, так как продукты разных производителей могут иметь одинаковые имена, а следовательно, не могут быть установлены на одну машину.

Для решения этой проблемы компания Microsoft разработала идею нумерации компонентов уникальными номерами. Этот номер называется GUID (Globally Unique Identifier), что означает глобально уникальный идентификатор. GUID представляет собой 16-байтное число, вычисленное специальной хэш-функцией, учитывающей следующие данные:

- текущие дату и время;
- «тики таймера»;
- простой инкрементный счетчик, чтобы можно было бы правильно обрабатывать следующие очень часто запросы на генерацию GUID;
- истинно глобально уникальный IEEE-идентификатор машины, извлекаемый из сетевой карты. Если в машине нет сетевой карты,

идентификатор может быть синтезирован из очень непостоянных машинных характеристик.

В современных средах разработки приложений, таких как Delphi, Visual Studio, реализованы средства автоматического получения уникального номера через API-функцию CoCreateGuid.

Примечание 1.1. *Чтобы получить GUID в Delphi, надо нажать комбинацию клавиш Ctrl+Shift+G. Для автоматической генерации глобального уникального номера в программе используется процедура CoCreateGuid*

В программе GUID определяется как структура типа

```
TGUID=Record
        D1: LongWord;
        D2: Word;
        D3: Word;
        D4: array [0..7] of Byte;
      End;
```

Четыре поля TGUID соответствуют четырем составным 16-байтного числа. Для удобства это число может быть записано в следующей символьной шестнадцатеричной форме (каждая шестнадцатеричная цифра заменена символом X):

‘{XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX}’

Примеры записи GUID:

‘{168A8BE0-485B-11D5-B5D2-444553540000}’

‘{168A8BE1-485B-11D5-B5D2-444553540000}’

‘{168A8BE2-485B-11D5-B5D2-444553540000}’

Для преобразования символьного представления GUID в двоичное и обратно в Delphi реализованы следующие системные функции:

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

Из алгоритма получения GUID и размерности числа видно, что этот номер действительно уникальный. Каждый компонент должен иметь свой уникальный номер. Этот номер называют CLSID (Class Identifier), что означает идентификатор класса.

Примечание 1.2. *Когда используется GUID, составленный из очень маленьких чисел, это, как правило, является признаком принадлежности компонента к ядру, разработанному Microsoft. GUID, сгенерированные*

другими производителями, будут выглядеть более случайными и не будут иметь столько нулей.

1.4. Доступ к компонентам. Интерфейсы

Фактически реализация компонента закрыта для приложения. Пользователя не должны интересовать способы реализации компонента, ему необходимо знать только функциональные возможности, которые может предоставить компонент. Доступ к компонентам осуществляется через интерфейс, который играет роль контракта (договора) между объектом и пользователем. По существу интерфейс представляет собой список указателей на процедуры и функции, реализованные в компоненте.

Таким образом, интерфейс похож на абстрактный класс, в котором перечислены виртуальные процедуры и функции, реализация которых ложится на дочерние классы. Иными словами, интерфейс определяет поведение объекта, но не поддерживает средства реализации этого поведения.

Еще одним важным свойством COM-компонентов является возможность поддержки сразу нескольких интерфейсов. Это дает возможность разбить большой набор процедур на несколько небольших логических групп.

Примечание 1.3. *Реализованные и используемые интерфейсы не меняются никогда. При модификации компонента существующий интерфейс не меняется, а добавляется новый. Это обеспечивает контекстную независимость COM.*

Как и компоненты, интерфейсы тоже имеют свой номер, который называют IID (Interface Identifier), то есть идентификатор интерфейса. Это позволяет использовать интерфейсы, не опасаясь совпадения имен.

1.5. Иерархия интерфейсов. Интерфейс IUnknown

В основе всех COM-интерфейсов лежит интерфейс с именем IUnknown. Этот интерфейс поддерживает три важные функции, которые наследуются каждому интерфейсу:

```
IUnknown=interface
      ['{00000000-0000-0000-C000-000000000046}']
    function QueryInterface(const IID: TGUID; out obj): HResult;
    stdcall;
    function _AddRef: Integer; stdcall;
    function _Release: Integer; stdcall;
end;
```


При динамической загрузке компонента система не знает, какие поддерживаются интерфейсы, но знает, что прямо или косвенно поддерживается интерфейс IUnknown, в котором реализована функция QueryInterface со следующими параметрами:

IID – идентификатор запрашиваемого интерфейса;

Obj – предназначен для хранения возвращаемого интерфейсного указателя.

Функция QueryInterface должна быть реализована в компоненте таким образом, чтобы можно было получить любой поддерживаемый интерфейс.

Если запрашиваемый интерфейс в компоненте реализован, QueryInterface возвращает указатель на интерфейс в параметре obj и значение S_OK, иначе – E_NOINTERFACE.

После работы с компонентом необходимо его выгрузить из памяти. Сложности могут возникнуть, когда на компонент, вернее, на его интерфейсы, имеется несколько ссылок. В больших проектах уследить за этим практически невозможно.

Чтобы избежать такой ситуации используется счетчик ссылок. Когда клиент получает некоторый интерфейс, значение этого счетчика увеличивается на единицу. Когда клиент заканчивает работу с интерфейсом, значение на единицу уменьшается. Когда оно доходит до нуля, компонент удаляет себя из памяти. Клиент также увеличивает счетчик ссылок, когда создает новую ссылку на уже имеющийся у него интерфейс. Для проведения этих операций предназначены функции _AddRef и _Release.

_AddRef – увеличивает счетчик ссылок на единицу и возвращает количество ссылок.

_Release – уменьшает счетчик ссылок на единицу и возвращает количество ссылок. Если счетчик равен нулю, уничтожает объект.

Для корректной работы приложения и компонентов необходимо строго соблюдать три правила, принятых при работе с COM:

1. *Функции, возвращающие интерфейсы, перед возвратом всегда должны вызвать _AddRef для соответствующего указателя. Это также относится и к QueryInterface.*
2. *После завершения работы с интерфейсом следует вызвать для него _Release.*
3. *Всегда следует вызывать _AddRef, когда создается новая ссылка на данный интерфейс.*

Приведем пример реализации QueryInterface, _AddRef, _Release в некоем Компоненте TSomeObject:

```
type TSomeObject=class(TObject, ISomeInterface)
    private
```

```
        FRefCount: integer;
        <реализация закрытой части>
    public
        //описание интерфейса IUnknown
        function QueryInterface(const IID:TGUID; out obj):HResult;
        stdcall;
        function _AddRef: Integer; stdcall;
        function _Release: Integer; stdcall;

        <описание открытой части>

    end;
<…….>
function TSomeObject. QueryInterface(const IID:TGUID; out
obj):HResult;
begin
    if GetInterface(IID, Obj) then
    begin
        Result:=S_OK;
        _AddRef;
    end else Result:=E_NOINTERFACE;
end;

function TSomeObject._AddRef: Integer;
begin
    Inc(FRefCount);
    Result:=FRefCount;
end;

function TSomeObject._Release: Integer;
begin
    Dec(FRefCount);
    Result:=FRefCount;
    if FRefCount=0 then destroy;
end;

<…….>
```

В примере описан объект TSomeObject, поддерживающий интерфейс ISomeInterface, и показана реализация функций QueryInterface, _AddRef и _Release.

1.6. Портативность и коммуникабельность COM. Регистрация COM-серверов. Системный реестр Windows

Ещё одно главное правило COM-технологии – отсутствие контекстной зависимости клиента и сервера. Под контекстной независимостью подразумевается, что клиент должен получить связь с сервером, где бы он ни находился.

Связь клиента с сервером берёт на себя операционная система, которая содержит системную базу данных, называемую *системным реестром*, где прописывается название COM-сервера, уникальный номер компонента CLSID и другая информация.

Примечание 1.4. *Реестр содержит информацию об аппаратном и программном обеспечении, о конфигурации компьютера и о пользователях. Любая программа для Windows может добавлять и считывать информацию из реестра.*

Реестр имеет иерархическую структуру в виде дерева, где узлами являются разделы, а листьями – данные.

Для просмотра и редактирования реестра можно использовать стандартную программу regedit.exe – редактор реестра (рис. 1.3).

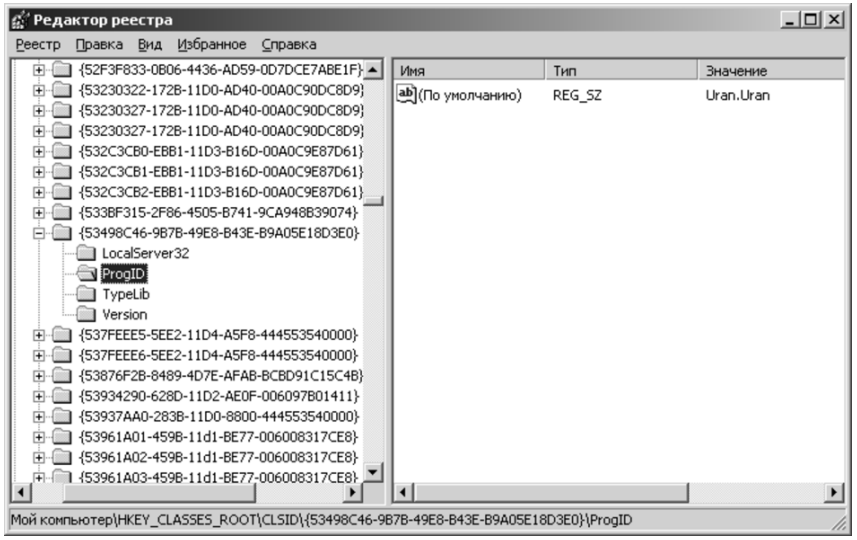

Рис. 1.3. Редактор реестра

Примечание 1.5. *Системный реестр является очень важной сущностью в операционной системе. Удаление или изменение в нём каких-либо данных может привести к невозможности работы некоторых приложений, а то и всей операционной системы. Все изменения в реестре записываются сразу на диск, откатов в системе нет. Удаление или изменение данных в системном реестре без четкого понимания сути происходящего крайне опасно. Попытки феноменологического исследования "что будет, если удалить вот это…" могут плохо кончиться.*

Данные об объектах COM хранятся в подразделе CLSID раздела HKEY_CLASSES_ROOT. В этом разделе перечислены CLSID всех компонентов, установленных в системе. Каждый CLSID содержит параметр по умолчанию, называемый *«дружественным» именем* компонента. В разделе описания компонента есть подраздел LocalServer32 (для внешних COM-серверов) или InprocServer32 (для внутренних COM-серверов), содержащих имя приложения или DLL, в которых находится компонент.

Имя файла и CLSID – это наиболее важные данные для нормального функционирования, но для некоторых, более сложных компонентов COM, хранится и другая информация. Например, это номер библиотеки типов, поддерживаемой данным COM-сервером (TypeLib), номер версии COM-сервера (Version), программное имя (ProgID) и т. п.

Чтобы зарегистрировать встроенный COM-сервер в системном реестре, необходимо вызвать функцию DllRegisterServer, находящуюся внутри DLL-модуля COM-сервера. Это можно сделать в приложении, предварительно загрузив DLL, или с помощью стандартной утилиты Windows RegSvr32.exe, выполнив команду:

RegSvr32 -s <имя сервера>

При удалении встроенного сервера из системы необходимо запустить функцию DllUnregisterServer. В Windows удаление сервера осуществляется командой:

RegSvr32 -u <имя сервера>

Для регистрации внешних COM-серверов требуется запустить приложение, включив в командную строку запуска ключ /regserver, а для удаления нужно использовать ключ /unregserver. Встроенный COM-сервер регистрируется и при первом запуске приложения без всяких дополнительных ключей в командной строке, но после этого сервер будет продолжать работать.

II. Разработка COM-серверов в Delphi

2.1. Разработка интерфейса

Прежде, чем реализовывать COM-сервер, нужно разработать интерфейс. Интерфейс является связью между клиентом и сервером (п. 1.4).

В Delphi интерфейсы описываются в программном модуле по следующей структуре:

<имя интерфейса>=**interface**
[<строка с IID>]
{список процедур и функций}
end;

Интерфейсы, как и классы, обладают свойством наследования:

<имя интерфейса>=**interface**(<имя родительского интерфейса>)
[<строка с IID>]
{список процедур и функций}
end;

Примечание 2.1. *В технологии COM принято название интерфейсов начинать с префикса «I» (IUnknown, IDispatch, IMyInterface и т. п.).*

Примечание 2.2. *IID должен быть уникальным, поэтому, всякий раз разрабатывая новый интерфейс, используйте генератор уникальных идентификаторов системы (в Delphi это комбинация клавиш Ctrl+Shift+G).*

Примечание 2.3. *При описании методов интерфейсов необходимо использовать директиву stdcall, которая определяет порядок передачи параметров и вызова функций в соответствии со стандартом, принятым в операционной системе Windows.*

Все интерфейсы прямо или косвенно наследуются от Iunknown, который содержит важные функции для реализации технологии COM (п. 1.5).

Для корректной работы приложения в интерфейсе следует описывать функции, возвращающие данные через параметры, а в качестве результата – информацию о корректности выполнения операций. Это может быть условный код ошибки или же просто информация об успешности выполнения, например, true – функция выполнена успешно, false – возникли ошибки. Такая реализация дает возможность контролировать и обрабатывать ошибки в клиенте.

Клиент и сервер должны содержать одинаковое описание интерфейсов. Поэтому интерфейс лучше описать в отдельном модуле, который присоединяется к клиенту и к серверу.

Приведем пример реализации интерфейса. В данном примере описан интерфейс, содержащий три функции перевода целого числа в двоичный, восьмеричный и шестнадцатеричный вид, который будет выводиться в виде строки.

1. Запустим Delphi и откроем новый модуль, выбрав в меню File/New. В появившемся диалоговом окне New Items на закладке New выберем элемент Unit.
2. Сохраним новый модуль как ConvertInterface.pas и в него поместим следующий текст:

```
unit ConvertInterface;

interface

type
  IConvert=interface
    ['{F1E9A4BA-4C4A-11D5-8D36-006008159451}']
    function Bin (const n: Word; var Str: WideString): Boolean;
    stdcall;
    function Oct(const n: Word; var Str: WideString): Boolean;
    stdcall;
    function Hex(const n: Word; var Str: WideString): Boolean;
    stdcall;
  end;

implementation
end.
```

2.2. Реализация встроенного COM-сервера

Встроенные COM-серверы реализованы внутри модулей DLL, поэтому они выполняются в процессе приложения, которое их использует. Вследствие данного свойства они и получили свое название.

Для того, чтобы создать встроенный COM-сервер в среде Delphi, необходимо в главном меню выбрать File/New. В появившемся диалоговом окне New Items открыть закладку ActiveX (рис. 2.1) и выбрать элемент ActiveX Library.

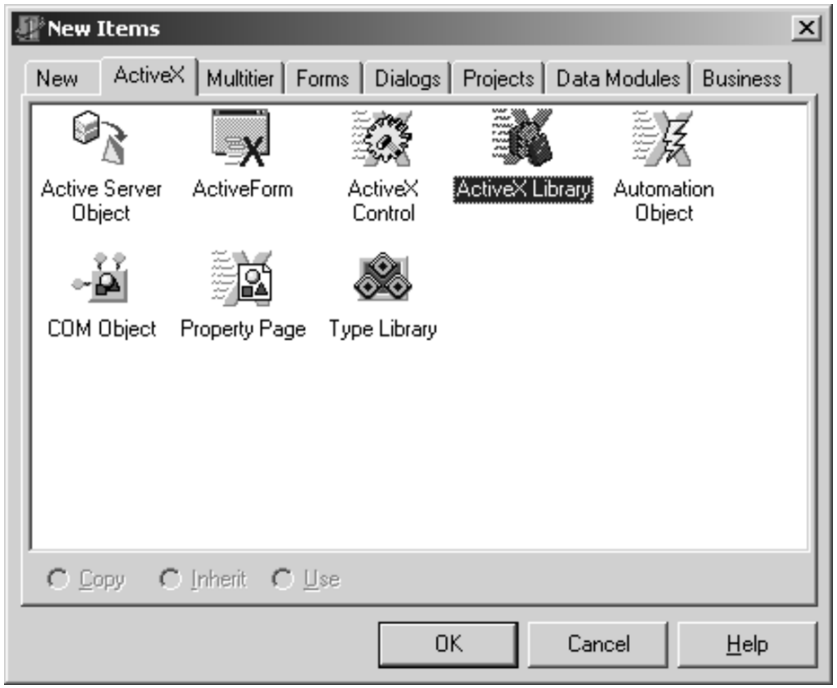

Рис. 2.1. Диалоговое окно New Items на закладке ActiveX

В библиотеку автоматически прописываются четыре экспортируемые функции, реализованные в модуле ComServ:

function DllRegisterServer: HResult; **stdcall**; – используется для регистрации COM-сервера в системном реестре. Если регистрация прошла успешно, функция возвращает значение S_OK, иначе S_FAIL.

function DllUnregisterServer: HResult; **stdcall**; – обратная функция к DllRegisterServer: удаляет из реестра Windows все элементы, касающиеся соответствующего компонента. Если функция отработала успешно, то возвращается значение S_OK, иначе S_FAIL.

function DllGetClassObject(**const** CLSID, IID: TGUID; **var** Obj): HResult; **stdcall**; – Создает новый компонент с номером CLSID и выдает ссылку на интерфейс с номером IID через параметр Obj. Вызывается системой, реализующей архитектуру COM. В случае успешного выполнения функция возвращает значение: S_OK иначе: E_NOINTERFACE – если интерфейс не найден и CLASS_E_CLASSNOTAVAILABLE – компонент с таким номером не зарегистрирован в системе.

function DllCanUnloadNow: HResult; **stdcall**; – вызывается механизмом реализации COM для проверки, выполняются ли условия удаления

COM-сервера из памяти. Если какое-либо приложение имеет ссылку на любой компонент сервера, то функция вернет S_FALSE, в противном случае – S_TRUE. В последнем случае механизм реализации COM удалит COM-сервер из памяти.

2.3. Разработка компонентов

Любой COM-сервер (встроенный или внешний) не имеет какого-либо смысла, если в нем не хранится хотя бы один компонент.

Чтобы в разрабатываемом COM-сервере создать новый компонент, надо выбрать в главном меню Delphi пункт File/New. Затем в появившемся диалоговом окне New Items (рис. 2.1) на закладке ActiveX выбрать элемент COM Object. После чего откроется окно мастера COM Object Wizard (рис. 2.2).

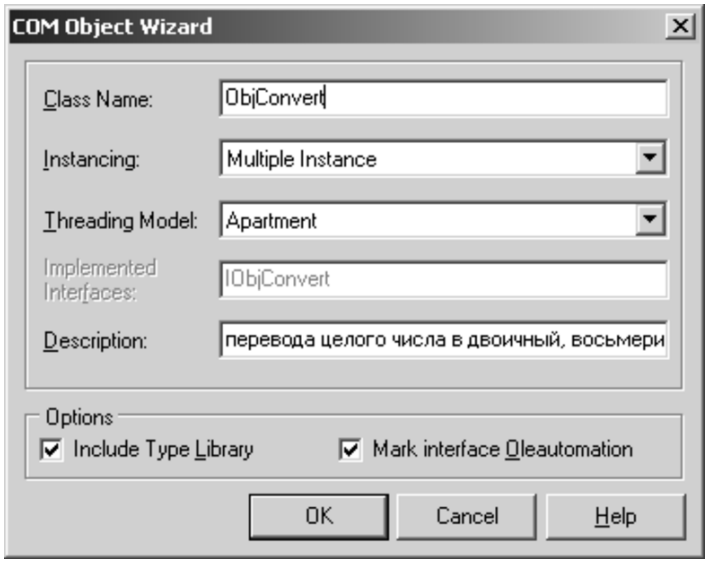

Рис. 2.2. Мастер COM Object Wizard

В поле Class Name вводится имя класса. Префикс «T» к названию объекта добавится автоматически. Название объекта не должно быть служебным словом и совпадать с именем COM-сервера и присоединенного модуля.

Поле Instancing определяет *метод создания экземпляра компонента* и актуально только для внешних COM-серверов:

Internal (внутренний) – используется только для компонентов, которые не предполагается сделать доступными для клиентских приложений. Запрашивать создание экземпляра такого компонента разрешено только собственному COM-серверу;

Single Instance (единственный экземпляр) – для каждого клиентского приложения допускается создание только одного экземпляра компонента. Приложение, запрашивающее компонент, будет работать с собственной отдельной копией COM-сервера;

Multiple Instance (множество экземпляров) – COM-сервер способен создавать множество копий компонента. При запросе нового экземпляра компонента новый сервер не запускается (если он уже запущен ранее), а создается новый экземпляр COM-объекта.

Threading model – определяет тип поддерживаемой *модели потоков задач*. Данный параметр актуален только к встроенным COM-серверам и может принимать следующие значения:

Single (однопотоковая) – все заявки на доступ к компоненту выстраиваются операционной системой Windows в очередь, так что нет необходимости беспокоиться о том, что несколько потоков могут одновременно иметь доступ к серверу;

Apartment (секционированная) – компоненты обрабатывают запросы только из того же потока, который их породил. Один сервер может экспортировать множество COM-объектов, причем каждый из них может быть создан в другом потоке, поэтому требуется синхронизация доступа к любым глобальным данным, которые определены в сервере посредством *мьютексов*, *событий*, *критических секций* и т. п.

Free (свободная) – множество потоков может быть активно в любом данном компоненте в одно и то же время. Вследствие этого нужно позаботиться о синхронизации доступа не только к глобальным данным, но и к локальным;

Both (и секционная и свободная) – компонент должен синхронизировать свой собственный доступ к экземплярам компонентов, настроенных на секционированную модель, и выполнять маршрутизацию параметров между потоками;

В поле Implemented interfaces через запятую вводятся названия интерфейсов, которые будет поддерживать компонент.

Поле Description предназначено для ввода замечаний или строки-описания объекта.

Когда выбирается опция Include Type Library, к описанию Компонента подключается библиотека типов. Библиотеки типов разобраны в третьей главе.

Компоненты в Delphi описываются так же, как и классы объектов:

```
<имя компонента>=class(<имя объекта-родителя>
{,<интерфейс}})
private
                <……….>
protected
                <методы интерфейсов>
                <……….>
public
                <……….>
end;
```

Для быстрого построения COM-приложений разработчики Delphi поместили в модуль ComObj классы, в которых реализованы все стандартные требования для COM. Так, например, есть класс TComObject, в котором реализованы все методы интерфейса IUnknown.

Примечание 2.4. *При реализации COM-приложений всегда пользуйтесь шаблонами, предложенными в закладке ActiveX диалогового окна New Items (рис. 2.1). Для каждого типа задач, основанных на COM, в Delphi реализованы свои классы, применение которых существенно облегчает работу и гарантирует корректность работы вашего проекта.*

Интерфейсы, поддерживаемые компонентом, описываются через запятую в разделе описания класса-предка, а методы интерфейса обычно описываются в разделе описания protected.

Примечание 2.5. *Если нужно инициализировать какие-либо данные во время создания компонента, то для этих целей используйте метод Initialize, который вызывается после работы конструктора.*

2.4. Разрешение неоднозначности методов

Как уже было неоднократно сказано, компонент может поддерживать несколько интерфейсов. Если методы разных интерфейсов с одинаковыми именами выполняют одинаковые функции, проблем не возникает, а когда требуется, чтобы они выполняли разные функции, используется *специальная форма выражения для разрешения методов*, пример реализации которых приведен в следующем примере:

```
IInterface1=interface
    procedure DoIt; stdcall;
end;
```



```
IInterface2=interface
    procedure DoIt; stdcall;
end;

TMyObject=class(TComObject, IInterface1, IInteface2)
    procedure IInterface1.DoIt=DoIt1;
    procedure IInterface2.DoIt=DoIt2;

    procedure DoIt1; stdcall;
    procedure DoIt2; stdcall;
end;
```

Класс TMyObject отображает метод IInterface1.DoIt на DoIt1, а метод IInterface2.DoIt – на DoIt2.

2.5. Делегирование интерфейса

Пусть имеется хорошо продуманный класс TObject1, в котором реализован интерфейс IInterfase1. Требуется создать класс TCombinedObject, в котором будут реализованы два интерфейса IInterfase1 и IInterfase2. Для этих целей в среде Delphi разработан *метод делегирования интерфейса*, основанный на том, что класс-преемник содержит указатель на другой класс, в котором уже реализовано поведение одного или более интерфейсов. Как можно осуществить делегирование, показано в следующем примере:

```
unit Unit1;

uses ComObj;

type
    IInterface1=interface
        procedure DoIt1;
    end;

    IInterface2=interface
        procedure DoIt2;
    end;

    TObject1=class(TComObject, IInterface1)
    protected
        procedure DoIt1;
    end;

    TCombinedObject=class(TComObject, IInterface1, IInterface2)
    private
```

```
        Fobj1: IInterface1;
    protected
        property Intf: IInterface1 read FObj1 implements IInterface1;

        procedure DoIt2;

        constructor Create;

        destructor Destroy;override;
    end;

implemetation

{TObject1}

procedure TObject1.DoIt1;
begin
    <……>
end;

constructor TCombinedObject.Create;
begin
    inherited;
    FObj1:=TObject1.Create;
end;

destructor TCombinedObject.Destroy;
begin
    FObj1:=nil;
    inherited;
end;

procedure TCombinedObject.DoIt2;
begin
    <……>
end;
```

Пусть в программе, использующей компонент TCombinedObject, есть переменная I2 – ссылка на интерфейс IInterface2. Чтобы получить ссылку на IInterface1, используется оператор as:

```
I1:=I2 as IInterface1;
```

Примечание 2.6. *Как было сказано в п. 1.5, для получения ссылки на другой поддерживаемый интерфейс нужно использовать функцию QueryInterface, но в Delphi данная операция реализована с помощью оператора as.*

Объект TObject1 данного примера принято называть *агрегированным* (объединяемым), а объект, который хранит ссылку на агрегируемый объект (TCombinedObject), называют *контейнером*.

Примечание 2.7. *Обратите внимание, что из интерфейса агрегированного объекта нельзя получить ссылку на контейнер. Иными словами, выражение I2 **as** Interface1 в нашем примере сгенерирует ошибку.*

2.6. Генератор компонентов

Для порождения нового экземпляра компонента, при запросе приложением-клиентом, используется стандартный механизм COM, названный *генератором компонентов,* или *фабрикой класса*. Любой компонент имеет ассоциированный с ним генератор компонентов, который поддерживает стандартный интерфейс IClassFactory:

```
IClassFactory = interface(IUnknown)
    ['{00000001-0000-0000-C000-000000000046}']
  function CreateInstance(const unkOuter: IUnknown; const iid: TIID;
                          out obj): HResult; stdcall;
  function LockServer(fLock: BOOL): HResult; stdcall;
end;
```

Функция CreateInstance ответственна за создание экземпляра компонента, на который ссылается генератор компонентов. Как правило, нет необходимости вызывать данную функцию самостоятельно, поэтому подробное описание параметров в данном методическом пособии рассматриваться не будет.

LockServer используется системой для подсчета ссылок обращений к COM-серверу. Когда приложение запрашивает компонент, в данную функцию передается параметр true и счетчик увеличивается на единицу. Когда COM-сервер не нужен приложению, в LockServer передается параметр false и счетчик на единицу уменьшается. В случае обнуления счетчика COM-сервер автоматически выгружается.

Примечание 2.8. *Microsoft объявила еще один интерфейс IClassFactory2, который добавляет к IClassFactory поддержку лицензирования или разрешения на создание. Клиент обязан передать генератору компонентов при помощи IClassFactory2 корректный ключ или лицензию, прежде чем этот генератор будет создавать компоненты. [4]*

Когда клиент передает системе COM запрос на новый экземпляр, сначала ищется и загружается (если он не загружен) COM-сервер, соответствующий запрашиваемому компоненту. Затем система COM подключается к фабрике компонентов через интерфейс COM, вызывает соответствующую функцию CreateInstance и передает клиенту соответствующую ссылку.

На первый взгляд, может возникнуть впечатление, что существует излишняя дуальность на наличие класса компонента и соответствующей ему фабрики класса, так как одна фабрика классов может взять на себя все заботы о создании множества компонентов. Но на самом деле имеется только один экземпляр генератора компонентов (ComServer), который находится в поле зрения COM и содержит список всех внутренних объектов фабрик классов.

Генератор компонентов изолирует механизм реализации COM от самого процесса формирования объекта. Если бы такого генератора не было, то пришлось бы выполнять непосредственный вызов конструктора класса, чтобы создать объект. Основная идея технологии состоит в том, чтобы не накладывать ограничений на то, как реализован объект, а выполняемое конструктором создание есть не что иное, как один из компонентов процесса реализации.

Примечание 2.9. *Следует заметить, что технология COM вовсе не требует, чтобы компоненты были реализованы с использованием ООП, просто такой подход является наиболее удобным.*

Помимо функций порождения компонентов, фабрика классов хранит данные о методе создания экземпляра компонента, тип модели потоков задач и другую необходимую информацию, которая передается в конструктор. Ниже приведен конструктор класса TClassFactory, используемый для реализации простых компонентов:

TClassFactory.Create(ComServer *{стандартное имя фабрики классов в Delphi}*, *<имя класса>*, *<CLSID класса>*, *<метод создания экземпляра компонента>, <тип модели потоков задач>*);

Метод создания экземпляра компонента и тип модели потоков задач автоматически устанавливаются на основе данных, введенных в мастере COM Object Wizard (рис. 2.2).

Для простоты обращения к фабрике класса используются Со-классы, которые поддерживают три функции:

```
<Имя Со-класса> = class
    class function Create: <интерфейс компонента>;
    class function CreateRemote(const MachineName: string): <интерфейс компонента>;
end;
```

Create – применяется для получения нового экземпляра компонента.

CreateRemote – создает экземпляр компонента на удаленной машине (применяется в DCOM).

Обычно, если планируется, что компонент будет запрашиваться клиентом, то для него создается Со-класс, чтобы не заучивать достаточно длинный уникальный номер CLSID компонента.

Примечание 2.10. *Обычно название Со-классов начинается с префикса «Со» и похоже на название компонента, который они порождают.*

2.7. Пример реализации встроенного COM-сервера

В качестве примера реализации встроенного COM-сервера создадим компонент перевода целого числа в двоичный, восьмеричный и шестнадцатеричный вид, который будет выводиться в виде строки. Данный компонент будет поддерживать интерфейс, приведенный в примере п. 2.1.

1. Выполнив действия, описанные в п. 2.2, сохраним проект под именем Convert.dpr.
2. Затем выберем элемент COM Object в диалоговом окне New Items (п. 2.3). В мастере COM Object Wizard отключим опции Include TypeLibrary и Mark Interfaced Oleautomation. Заполним соответствующие поля: в Class Name введем ObjConvert, а в Implemented interfaces – IConvert. Нажав на кнопку OK, получим модуль следующего содержания:

```
unit Unit1;

interface

uses
    Windows, ActiveX, ComObj;

type
    TObjConvert = class(TComObject, IConvert)
    protected
        {Declare IConvert methods here}
    end;

const
  Class_ObjConvert: TGUID = '{B65CC116-4C5F-11D5-8708-
  854E82B6237}';

implementation

uses ComServ;

initialization
```

```
    TComObjectFactory.Create(ComServer, TObjConvert,
            Class_ObjConvert, 'ObjConvert', 'Convert', ciSingleIn-
            stance, tmSingle);
end.
```

3. В разделе описания используемых модулей после Windows, ActiveX и ComObj следует добавить еще и модуль ConvertInterface, в котором реализован интерфейс IConvert.
4. В разделе описания типов дано начальное описание нашего компонента. Исходя из этого описания, наш компонент наследуется от класса TComObject, и поддерживает интерфейс IConvert. В разделе protected написан комментарий «Declare IConvert methods here» (опишите здесь методы IConvert). В этом разделе описываются все функции, которые поддерживаются интерфейсом:

```
TObjConvert = class(TComObject, IConvert)
protected
    {Declare IConvert methods here}
    function Bin (const n: Word; var Str: WideString): Boolean;
    stdcall;
    function Oct(const n: Word; var Str: WideString): Boolean;
    stdcall;
    function Hex(const n: Word; var Str: WideString): Boolean;
    stdcall;
end;
```

5. Остается реализовать функции объекта TObjConvert в разделе implementation:

```
implementation

uses ComServ;

{ TObjConvert }

function TObjConvert.Bin(const n: Word; var Str: WideString): Boo-
lean;

var
    S: String;
    m: Word;

begin
    result:=true;
    try
        S:='';
        m:=n;
```



```
        repeat
            S:=chr(m mod 2+48)+S;
            m:=m div 2;
        until m=0;
        Str:= WideString(S);
    except
        result:=false;
    end;
end;

function TObjConvert.Oct(const n: Word; var Str: WideString): Boo-
lean;

var
    S: String;
    m: Word;

begin
    result:=true;
    try
        S:='';
        m:=n;
        repeat
            S:=chr(m mod 8+48)+S;
            m:=m div 8;
        until m=0;
        Str:=WideString (S);
    except
        result:=false;
    end;
end;

function TObjConvert.Hex(const n: Word; var Str: WideString): Boo-
lean;

var
    S: String;
    m,k: Word;

begin
    result:=true;
    try
        S:='';
        m:=n;
        repeat
```

```
            k:=m mod 16;
            if k<10 then S:=chr(k+48)+S
                    else S:=chr(57+k)+S;
            m:=m div 16;
          until m=0;
          Str:=WideString(S);
        except
          result:=false;
        end;
      end;
```

6. Приложение можно откомпилировать, нажав сочетание клавиш Ctrl+F9. Созданный при этом COM-сервер, находящийся в библиотеке Convert.dll, готов к использованию.

В данном примере используются строки типа WideString. Это стандартный строковый тип, используемый в технологии COM. В простых строках (String, ShortString, PChar и т. п.) символы имеют размерность 8 бит (1 байт), а в WideString – 16 бит (2 байта). Это дает возможность передачи строк в кодировке Unicode.

2.8. Внешние COM-серверы

Внешние COM-серверы реализованы в виде программ (exe-файлов) и в процессе выполнения занимают адресное пространство, отличное от клиентского. Это приводит к некоторым нюансам при создании такого рода COM-серверов.

В отличие от встроенных, внешние COM-серверы не экспортируют функции DllRegisterServer, DllUnregisterServer, DllGetClassObject и DllCanUnloadNow.

Так как клиент и внешний COM-сервер не могут получить прямой доступ к адресному пространству друг друга, система COM передает данные между ними посредством процесса, называемого *маршалингом*[1].

Система COM, реализованная в операционной системе Windows, может передавать посредством маршалинга переменные следующих типов: Smallint, Integer, Single, Double, Currency, TdateTime, WideString, IDispatch, SCODE, WordBool, OleVariant, IUnknown, ShortInt, Byte, Word, UINT, int64, Largeint, SYSINT, SYSUINT, HResult, Pointer, SafeArray, PChar, PWideChar.

Следует отметить, что все перечисленные типы относятся к категории простых типов, так как маршалинг переменных сложных типов (массивы, записи, структуры и т. п.) система COM не поддерживает. Если все-таки требуется передать данные типа запись или массив, нужно самостоятельно запрограммировать поддержку механизма маршалинга для таких

[1] От слова Marshaling – выстраивание в определенном порядке.

структур. Для этого понадобится интерфейс IMarshal. В данном методическом пособии реализация маршалинга рассматриваться не будет.

Внешние COM-серверы, как и обычные приложения, можно закрыть самостоятельно, но если на компоненты сервера ссылаются клиенты, то система выдает соответствующее сообщение (рис. 2.3).

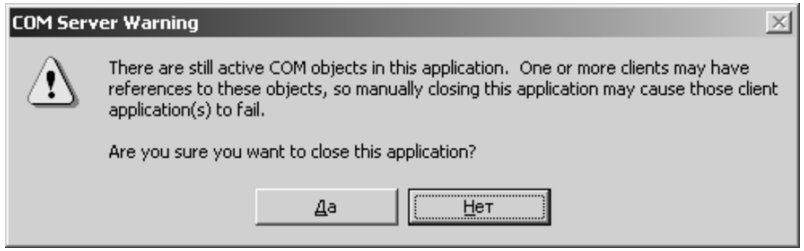

Рис. 2.3. Предупреждение, вызываемое при закрытии внешнего COM-клиента, если на компоненты сервера ссылаются клиенты

Пример внешнего COM-сервера будет приведен в пятой главе, так как для реализации таких серверов требуется библиотека типов, которая будет рассмотрена в гл. III.

2.9. Безопасные массивы (SafeArray)

Безопасные массивы (тип SafeArray) реализованы на основе вариантных массивов. Процедуры и функции для работы с вариантными массивами приведены в следующей таблице.

Подпрограмма	Описание
function VarArrayCreate(const Bounds: array of Integer; VarType: TVarType): Variant;	Создает вариантный массив. Параметр Bounds содержит размерность массива. Параметр VarType определяет тип хранимых данных
function VarArrayOf(const Values: array of Variant): Variant;	Создает одномерный вариантный массив из элементов Values. Индексация начинается с 0
function VarArrayRef(const A: Variant): Variant;	Возвращает ссылку на вариантный массив
function VarArrayDimCount(const A: Variant): Integer;	Возвращает количество измерений в вариантном массиве В случае, если это не массив, возвращает 0

function VarArrayLowBound(const A: Variant; Dim: Integer): Integer;	Возвращает нижнюю границу измерения Dim для массива
function VarArrayHighBound(const A: Variant; Dim: Integer): Integer;	Возвращает верхнюю границу измерения Dim для массива
function VarArrayLock(const A: Variant): Pointer;	Блокирует вариантный массив и возвращает ссылку на него
procedure VarArrayUnlock(var A: Variant);	Снимает блокировку с вариантного массива, заданную предыдущей функцией
function VarIsNull(const V: Variant): Boolean;	Возвращает истину, если вариант содержит пустое значение
procedure VarArrayRedim(A: Variant; HighBound: Integer);	Изменяет размер вариантного массива

В следующем пункте приведен пример программирования вариантных массивов.

2.10. Обработка массивов данных

Рассмотрим функциональные возможности безопасных массивов на примере.

2.10.1. Создание сервера

Как и в предыдущем примере, создадим внешний сервер автоматизации. В редакторе библиотеки типов определим два интерфейса – IvectSrv и IMatrixSrv. В первом интерфейсе создадим метод ChangeVect, выполняющий перестановку элементов массива. Тип первого параметра (исходный массив) определим как Variant с модификатором in, тип второго (результат вычислений) – Variant с модификатором out.

Во втором интерфейсе IMatrixSrv для транспонирования матрицы создадим метод Trans с аналогичными параметрами. Сгенерируем модуль с определением компонентного класса и запишем в него реализацию упомянутых методов:

```
type
  TVectSrv = class(TAutoObject, IVectSrv, IMatrixSrv)
  protected
    procedure ChangeVect(v:OleVariant; out z:OleVariant);  safecall;
```



```
    procedure Trans(a: OleVariant; out b: OleVariant);  safecall;
  end;

implementation

uses ComServ, variants;

procedure TVectSrv.ChangeVect(v: OleVariant; out z: OleVariant);
var dim, i,j: integer;
begin
  dim := VarArrayDimCount(v);  // число измерений
  if dim = 1 then
  begin
    z := v;
      // Определяем размерности по X переданного массива
    i := VarArrayLowBound(v,1);
    j := VarArrayHighBound(v,1);
    // Меняем местами первый и последний элемент
    z[i]:=v[j];
    z[j]:=v[i];
  end;
end;

procedure TVectSrv.Trans(a: OleVariant; out b: OleVariant);
var dim, n, m, i0,j0,i, j: integer;
begin
  b := a;
  dim := VarArrayDimCount(a);  // число измерений
  if dim = 2 then
  begin
    // Определяем размерности по X переданного массива
    i0 := VarArrayLowBound(a,1);
    n := VarArrayHighBound(a,1);
      // .. и размерности по Y
    j0 := VarArrayLowBound(a,2);
    m := VarArrayHighBound(a,2);

    // Транспонируем
    for i := i0 to n do
      for j := j0 to m do
        b[i,j] := a[j,i];
  end;
end;
```

2.10.2. Создание клиента

```
type
  TMainForm = class(TForm)
    lb: TListBox;
    bCreate: TButton;
    bRun: TButton;
    sg: TStringGrid;
    procedure bCreateClick(Sender: TObject);
   procedure bRunClick(Sender: TObject);
  private
   { Private declarations }
  public
   v:  IVectSrv;
   vm: IMatrixSrv;
   { Public declarations }
  end;

var
  MainForm: TMainForm;

implementation

var  X, Y, A , B: olevariant; // Нужны для передачи в COM сервер
    DynArr : array of array of integer;
                    // динамический двумерный массив

{$R *.dfm}

procedure TMainForm.bCreateClick(Sender: TObject);
var
  I, J: Integer;
begin
    // Для иллюстрации здесь применены два способа.
    //  Одномерный массив создается как вариантный.
    //  Матрица - как динамический двумерный массив.
    //  Задаем длину 2 на 2 двумерного динамического массива
  SetLength (DynArr,2,2);
    // .. и заполняем (лучше это сделать по-другому - из файла, списка или
сетки)
  DynArr[0,0]:=1;
  DynArr[1,1]:=3;
```



```
  DynArr[0,1]:=4;
  DynArr[1,0]:=2;
   // Выводим в сетку
  sg.ColCount := 2;
  sg.RowCount := 2;
  with sg do
    for I := 0 to ColCount - 1 do
      for J:= 0 to RowCount - 1 do
        Cells[J,I] := IntToStr(DynArr[I,J]);
       //  Создаем одномерный вариантный массив
  X := VarArrayOf ([1.2, 2.5, 3, 4.8]);
     // Выводим в ListBox
  lb.Clear;
  for i:=0 to VarArrayHighBound(X,1) do
   lb.Items.Add(FloatToStr(X[i]));
   //  Запускаем сервер
  V := CreateComObject(Class_VectSrv) as IVectSrv;
  if V=Nil then
    ShowMessage ('Сервер не найден!');
end;

procedure TMainForm.bRunClick(Sender: TObject);
var
  I, J: Integer;
begin
  if V <> Nil then
  begin
    try
          // Вызываем метод интерфейса (меняет местами элементы)
      V.ChangeVect(x,y);

          // Выводим результат
      lb.Clear;
      for i:=0 to VarArrayHighBound(Y,1) do
        lb.Items.Add(FloatToStr(Y[i]));
      Vm := V as IMatrixSrv;  // Обращаемся к другому  интерфейсу
      A := DynArr;     // Записываем в вариант динамический массив
      Vm.Trans(A,B);    // Вызываем метод (транспонирование)

      // Выводим в сетку
      with SG do
      for I := 0 to ColCount - 1 do
       for J:= 0 to RowCount - 1 do
```

```
          Cells[J,I] := IntToStr(B[I,J]);
      except
      on EOleException do
        ShowMessage ('Com сервер недоступен!');
      else
        ShowMessage ('Что-то не так  ...');
     end;
   end;
end;

end.
```

III. Библиотека типов

COM-серверы и клиенты, которые взаимодействуют друг с другом, должны поддерживать одинаковое описание интерфейсов и правила передачи параметров. Когда клиенты и серверы проектируются на одном языке программирования, этой проблемы не возникает, так как можно присоединить один и тот же модуль объявления интерфейсов. Чтобы работать независимо от языка программирования, было разработано стандартное описание интерфейса, которое называют библиотекой типов. Библиотека типов независима от языка программирования, и в проект она может быть внесена в виде ресурса с расширением tlb.

3.1. Проектирование библиотеки типов. Редактор Type Library Editor

Для того, чтобы библиотеку типов подключить к разрабатываемому COM-серверу автоматически, нужно в окне мастера COM Object Wizard (рис. 2.2) выставить флажок Include Type Library.

Для описания данных в библиотеке типов используется специальный язык IDL (Interface Definition Language – язык определения интерфейсов). Для автоматизации работы с библиотекой типов в Delphi разработан специальный редактор Type Library Editor (рис. 3.1), который можно запустить, выбрав в главном меню Vew/Type Library Editor.

Окно Type Library Editor разделено на три части. Вверху расположена панель инструментов, с помощью которой можно включать в COM-сервер новые интерфейсы, методы и свойства. Слева расположено поле списка объектов, в котором представлены данные в древовидной структуре. Справа – панель с закладками, в которых редактируется элемент, выбранный в поле списка объектов.

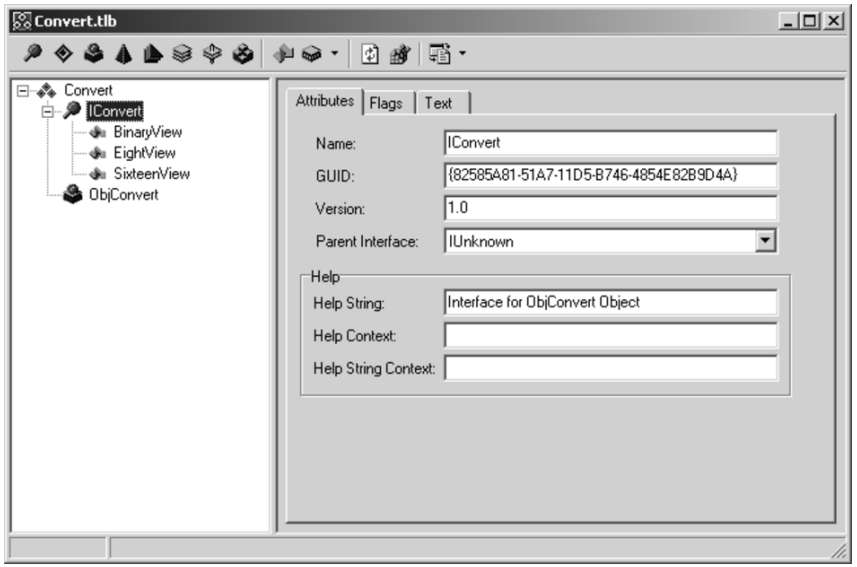

Рис. 3.1. Редактор Type Library Editor

3.2. Панель инструментов диалогового окна Type Library Editor

Рис. 3.2. Панель инструментов редактора Type Library Editor

Для добавления новых интерфейсов, свойств и методов используется панель инструментов (рис. 3.2), которая содержит следующие функциональные кнопки (слева направо):

New Interface – после щелчка на этой кнопке будет добавлен новый интерфейс;

New Dispinterface – добавляет новый интерфейс диспетчеризации. *Интерфейс диспетчеризации* (dispinterface) во многом схож с обычным интерфейсом, но использует отличный от него механизм диспетчеризации для вызова методов сервера (см. гл. V);

New CoClass – добавляет сопряженный класс. *Сопряженный класс* (CoClass) – это класс компонента, в котором реализован интерфейс (или интерфейсы);

New Enumeration – добавляет перечисление. *Перечисление* (enumeration) в библиотеке типов аналогично перечислимому типу в Delphi: объявляет набор целых идентификаторов;

New Alias – создает псевдоним. *Псевдоним* (alias) используется в библиотеке типов для спецификации типа, который желательно включить в состав записи или объединения (union);

New Record – задает структуру записи;

New Union – задает тип объединения. *Объединение* (union) – это эквивалент массива вариантного типа (array of variant);

New Module – определяет новый модуль. Модуль представляет собой набор методов и констант;

New Method – добавляет новый метод в интерфейс. Если в списке объектов выделить интерфейс, то эта кнопка становится активной;

New Property – добавляет новое свойство в интерфейс. Так же, как и в обычный класс Delphi, в интерфейс можно добавить свойство, которое будет *только для записи* (read-only), *только для чтения* (write-only) или *для чтения и записи* (read-write) одновременно;

Refresh Implementation – при нажатии на эту кнопку обновляется программный код приложения соответственно данным в редакторе Type Library Editor;

Register Type Library – после щелчка на этой кнопке Delphi запустит процесс компиляции создаваемого COM-сервера и зарегистрирует его в системном реестре Windows;

Export to IDL – с помощью этой кнопки можно сформировать программный код в формате MIDL или CORBA.

3.3. Использование библиотеки типов при разработке компонентов

Как уже было сказано, для подключения библиотеки типов к создаваемому COM-серверу нужно в мастере COM Object Wizard (рис. 2.2) выбрать флажок Include Type Library. Также можно подключить библиотеку типов вручную, добавив ее в проект в виде ресурса:

{$R MyTypeLib.tlb}

Примечание 3.1. *Иногда необходимо, чтобы несколько COM-серверов имели одинаковый интерфейс и были установлены на один компьютер одновременно. Для этих целей библиотека типов регистрируется в системе отдельно от COM-серверов.*

Когда используется библиотека типов, создаваемый компонент наследуется не от TComObject, а от TTypedComObject. И вместо генератора компонентов TComObjectFactory используется TTypedComObjectFactory.

TTypedComObject и TTypedComObjectFactory предназначены для работы с библиотекой типов и аналогичны классам TComObject и TComObjectFactory.

Для реализации COM-сервера в среде Delphi редактор Type Library Editor генерирует программный модуль на языке Object Pascal, непосредственно связанный с библиотекой типов. Если в редакторе Type Library Editor нажать на кнопку Refresh Implementation, то данные этого модуля обновятся соответственно данным в библиотеке (tlb).

3.4. Пример разработки встроенного COM-сервера с использованием библиотеки типов

Во второй главе был разобран пример разработки COM-сервера. Реализуем этот пример с использованием библиотеки типов.

1. Создадим встроенный COM-сервер, выбрав в диалоговом окне New Items (рис. 2.1) элемент ActiveX Library, и сохраним его под именем Convert.dpr.
2. Создадим новый Компонент, выбрав элемент COM Object. В Мастере COM Object Wizard (рис. 2.2) поставим флажок Include Type Library и уберем Mark interface Oleautomation (т. к. это будет рассмотрено в гл. V). В поле Class Name напишем ObjConvert.
3. Когда будет нажата кнопка OK, автоматически откроется окно редактора Type Library Editor. В поле списка объектов будут два элемента: ObjConvert и IObjConvert. Если интерфейса IObjConvert нет, то создайте его с помощью панели инструментов редактора.
4. У интерфейса IObjConvert создадим новый метод и назовем его Bin. На закладке Parameters в строке Return Type установим возвращаемый тип WordBool. В таблице добавим новый параметр, нажав на кнопку Add, и в поле Name укажем имя N, а в поле Type – Word. Аналогично добавим параметр Str, нажав на кнопку Add и выбрав в поле Modifier значение out и Retval, в поле Name – Str, а в поле Type – WideString.
5. Аналогично пункту 5 задаются методы Oct и Hex.
6. Нажав на кнопку Refresh Implementation, обновим реализацию кода.
7. Когда вернемся в редактор Delphi, модуль ObjConvertUnit будет содержать следующий текст:

```
unit ObjConvertUnit;

interface

uses
    Windows, ActiveX, ComObj, Convert_TLB;
```

```
type
    TObjConvert = class(TTypedComObject, IConvert)
    protected
        function Bin(N: Word; var Str: WideString): WordBool;
        stdcall;
        function Oct(N: Word; var Str: WideString): WordBool;
        stdcall;
        function Hex(N: Word; var Str: WideString): WordBool;
        stdcall;
        {Declare IObjConvert methods here}
    end;

implementation

uses ComServ;

function TObjConvert.Bin (N: Word; var Str: WideString): Word-
Bool;
begin

end;

function TObjConvert.Oct(N: Word; var Str: WideString): Word-
Bool;
begin

end;

function TObjConvert.Hex(N: Word; var Str: WideString): Word-
Bool;
begin

end;

initialization
  TTypedComObjectFactory.Create(ComServer, TObjConvert,
                    Class_ObjConvert, ciMultiInstance,
                    tmSingle);
end.
```

Это шаблон нашего COM-класса, подготовленный средой Delphi, который необходимо доработать.

8. В модуле ObjConvertUnit реализуем функции Bin, Oct, Hex, как в четвертом пункте второй части.
9. Приложение можно откомпилировать, нажав сочетание клавиш Ctrl+F9. Созданный при этом COM-сервер готов к использованию.

IV. Разработка приложений, использующих COM-серверы

4.1. Получение ссылки на интерфейс компонента. Функции CreateComObject и ProgIdToClassId

При работе с компонентом COM приложение должно прямо или косвенно получить ссылку на его интерфейс. Для этих целей в Delphi определена функция CreateComObject, которая в качестве параметра принимает CLSID запрашиваемого компонента, и, если компонент зарегистрирован в системе, возвращает интерфейс IUnknown, через который можно получить остальные интерфейсы:

```
function CreateComObject(const ClassID: TGUID): IUnknown;
```

Доступ к интерфейсам объектов автоматизации (см. гл. V) можно получить также и с помощью функции:

```
function CreateOleObject(const ClassName: string): IDispatch;
```

В качестве параметра в данную функцию передается дружественное имя, состоящее из имени COM-сервера и имени компонента, которые разделены точкой:

‘<имя COM-интерфейса>.<имя компонента>’

CLSID компонента можно получить с помощью функции ProgIdToClassId:

```
function ProgIDToClassID(const ProgID: string): TGUID;
```

В качестве параметра в функцию подается строка с дружественным именем компонента. Результатом является CLSID компонента, хранящегося в COM-сервере. Например, если вызвать данную функцию с параметром ‘Word.Application’, то функция вернет следующий GUID (идентификатор приложения Microsoft Word): ‘{000209FF-0000-0000-C000-000000000046}’.

Для того, чтобы получить указатель на уже запущенный в системе объект автоматизации, используется функция

```
function GetActiveOleObject(const ClassName: string): IDispatch;
```

Как и в процедуре CreateOleObject, в функцию передается дружественное имя объекта. Функция просматривает *таблицу выполняемых объектов* Windows (Running Table Objects) в поисках запрашиваемого сервера. Если такой будет найден, то функция вернет ссылку на интерфейс IDispatch этого сервера.

4.2. Экспорт описания интерфейсов из библиотеки типов COM-сервера

В третьей главе разобрано применение библиотеки типов в COM-серверах. Библиотека типов хранится в COM-сервере в виде ресурса, и доступ к ее данным можно получить с помощью стандартных средств WinAPI (подробнее см. [7 гл. 3]). Также в Delphi предусмотрено другое стандартное средство для генерации программного кода на языке Object Pascal, соответствующего описанию библиотеки типов. Для этого нужно открыть диалоговое окно Import Type Library (рис. 4.1), выбрав в главном меню элемент Project/Import Type Library….

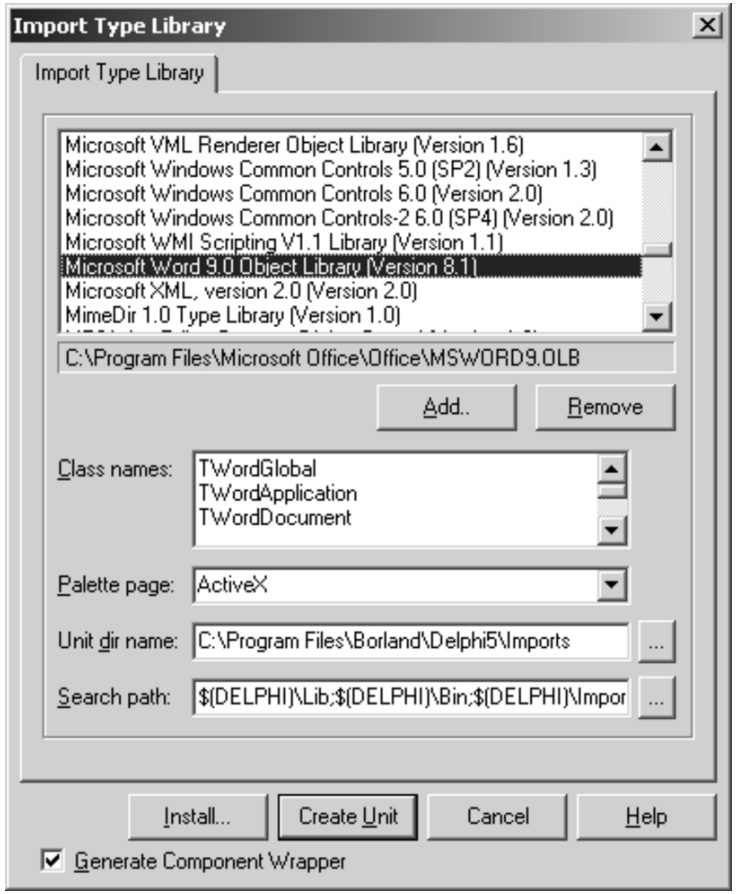

Рис. 4.1. Диалоговое окно Import Type Library

Вверху этого диалогового окна находится список существующих COM-серверов. Добавить новый COM-сервер в этот список можно с помощью кнопки Add…, удалить – Remove. Данные кнопки аналогичны командам регистрации и удаления COM-серверов в системе.

Ниже списка COM-серверов находится список компонентов в выбранном сервере, а также строка с выбором директории, в которой будет сохранен экспортируемый модуль.

Чтобы получить программный код, аналогичный описанию библиотеки типов, нужно нажать на кнопку Create Unit.

Кнопка Install… автоматически генерирует программный код компонента Delphi, в котором реализованы все свойства и методы выбранного в списке компонента. Полученный компонент устанавливается на закладке Server. Работа с такими компонентами тривиальна и аналогична, как и со всеми стандартными компонентами Delphi, поэтому данный метод рассматриваться не будет.

4.3. Особенности техники использования компонентов в Delphi

Delphi является такой средой разработки программ, в которой очень многое автоматизировано и упрощено. Не исключением является и COM-технология. В Delphi разработаны классы, которые существенно облегчают реализацию различных задач технологии COM:

Базовый класс компонента	Генератор компонентов	Используется для проектирования …
TAutoObject	TAutoObjectFactory	компонентов автоматизации (гл. V)
TComObject	TComObjectFactory	простых компонентов
TTypedComObject	TTypedComObjectFactory	компонентов с генерацией библиотеки типов

Еще одной важной особенностью разработки COM-приложений является то, что нет необходимости подсчитывать ссылки, то есть использовать методы _AddRef и _Release, так как система сама автоматически увеличит счетчик, когда ссылка будет присвоена, и уменьшит, когда переменной будет присвоено значение nil. Когда счетчик ссылок будет равен нулю, объект будет уничтожен:

V:=CreateComObject(GUID); *//счетчик ссылок увеличился и равен 1*

W:=V; *//счетчик ссылок увеличился и равен 2*

<……>

V:=**nil**; *//счетчик ссылок уменьшился и равен 1*
W:=**nil**; *//счетчик ссылок уменьшился, и объект удалился*

Замечание 4.1. *Всегда присваивайте ссылкам на компонент значение nil, чтобы уменьшить счетчик ссылок и своевременно выгрузить компонент из памяти.*

Так же, как и в классах, можно получить интерфейс объекта с помощью оператора **as**:

```
V:=CreateComObject(GUID);
W:=V as ISomeInterface; //аналогично V.QueryInterface(IID,W)
```

В пакете Delphi поставляется специальная утилита для регистрации COM-серверов и библиотеки типов tregsvr.exe. Обычно эта утилита находится в папке Delphi\Bin.

4.4. Пример реализации COM-клиента

Приведем пример использования встроенного COM-сервера, созданного в третьей части.

1. Создайте встроенный COM-сервер из примера в третьей части. Зарегистрируйте его, как написано в первой части шестого пункта.
2. Откройте новый проект в Delphi.
3. Откройте диалоговое окно импорта библиотеки типов (рис. 4.1) и найдите библиотеку нашего COM-сервера. Если COM-сервер называется Convert.dll, то в списке будет строка Convert Library (Version 1.0). Импортируйте ее.
4. На форму поставьте строку ввода TEdit и три кнопки Bin, Oct, Hex (рис. 4.2).

Рис. 4.2. Форма примера

5. В соответствующем форме модуле наберите следующий текст:

```
unit Unit1;

interface

uses
    Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs, StdCtrls, Convert_TLB;

type
    TForm1 = class(TForm)
        Edit1: TEdit;
        Bin: TButton;
        Oct: TButton;
        Hex: TButton;
        procedure FormCreate(Sender: TObject);
        procedure BinClick(Sender: TObject);
        procedure OctClick(Sender: TObject);
        procedure HexClick(Sender: TObject);
    private
        { Private declarations }
        Convert: IConvert;
    public
        { Public declarations }
    end;

var
    Form1: TForm1;

implementation

{$R *.DFM}

procedure TForm1.FormCreate(Sender: TObject);
begin
    //создается экземпляр Компонента
    Convert:=nil;
    Convert:=CoObjConvert.Create;
    if not Assigned(Convert) then
    begin
        ShowMessage('Не возможно создать компонент');
        Close;
    end;
end;

procedure TForm1.BinClick(Sender: TObject);

var str: WideString;

begin
    if Convert.Bin(StrToInt(Edit1.Text),Str) then ShowMessage(Str)
```

```
                else raise Exception.Create('Ошибка в BinClick');
end;

procedure TForm1.OctClick(Sender: TObject);

var str: WideString;

begin
    if Convert.Hex(StrToInt(Edit1.Text),Str) then ShowMessage(Str)
                else raise Exception.Create('Ошибка в OctClick');
end;

procedure TForm1.HexClick(Sender: TObject);

var str: WideString

begin
    if Convert.Hex(StrToInt(Edit1.Text),Str) then ShowMessage(Str)
                else raise Exception.Create('Ошибка в HexClick');
end;

procedure TForm1.FormClose(Sender: TObject; var Action: TCloseAction);
begin
    //разрыв с ссылкой. Счетчик уменьшается на единицу
    Convert:=nil;
end;

end.
```

V. Автоматизация

Автоматизация (*Automation*) представляет собой механизм автоматического управления одним приложением из другого. Существуют как встроенные серверы автоматизации, так и внешние.

В среде Delphi легко создавать и использовать серверы автоматизации, имея весьма слабое представление о принципах COM-технологии.

5.1. Раннее и позднее связывание. Тип Variant

Чтобы получить доступ к интерфейсу компонента автоматизации необязательно использовать описание интерфейса. В Delphi есть возможность использовать интерфейс без его описания. В этом случае ссылка на интерфейс создаваемого компонента хранится в переменной типа Variant. В этом случае компилятор не проверяет корректность обращений к интерфейсу. Корректность проверяется только уже на стадии выполнения программы. Такой метод называют *методом позднего связывания*.

Ниже приведен пример программного кода, в котором организуется подключение к Microsoft Word и ему передается команда сформировать новый файл:

```
procedure TForm1.Button1.Click(sender: TObject);

var V: Variant;

begin
    V:=CreateOLEObject('Word.Basic');
    V.AppShow;
    V.FileNew;
    V.Insert('Автоматизация это просто!');
    V:=NULL;
end;
```

Примечание 5.1. *При обнулении типизированной ссылки на интерфейс используется nil, а при обнулении ссылки типа Variant – NULL.*

При позднем связывании компилятор Delphi принимает все без возражений и не выполняет контроль корректности типов, но при вызове несуществующих методов во время выполнения программы появляется сообщение об ошибке (рис. 5.1).

Рис. 5.1. Сообщение об ошибке при обращении к несуществующему методу сервера автоматизации

Метод работы с компонентом, когда используется описание его интерфейса, называют *методом раннего связывания*. В этом случае корректность работы вызова процедур проверяется на стадии компиляции программы.

Недостаток позднего связывания – замедленное обращение к компоненту, так как система COM пытается найти вызываемую процедуру или функцию по имени. Но в противовес недостаткам существуют ряд преимуществ данного метода:

1) иногда программы или макросы разрабатываются в среде, которая не поддерживает интерфейсов (например, Visual Basic) и т. п;

2) не требуется импортировать и подключать программный модуль с описанием интерфейсов и типов данных, применяемых компонентом;
3) в некоторых методах компонентов параметры определены по умолчанию, но при раннем связывании компилятор Delphi не допустит возможности «опустить» данные параметры. Например, пусть есть переменная Word, в которой хранится указатель на компонент Microsoft Word. Чтобы создать новый документ при позднем связывании, достаточно написать оператор:

```
Word.Documents.Add
```

При раннем связывании придется вызывать данную функцию с параметрами:

```
Word.Documents.Add(Template, NewTemplate, DocumentType, Visible).
```

5.2. Интерфейсы диспетчеризации

Механизм позднего связывания реализуется в самих компонентах автоматизации, которые обязаны поддерживать интерфейс IDispatch:

```
IDispatch = interface(IUnknown)
  ['{00020400-0000-0000-C000-000000000046}']
  function GetTypeInfoCount(out Count: Integer): HResult; stdcall;
  function GetTypeInfo(Index, LocaleID: Integer; out TypeInfo): HResult; stdcall;
  function GetIDsOfNames(const IID: TGUID; Names: Pointer; NameCount, LocaleID: Integer; DispIDs: Pointer): HResult; stdcall;
  function Invoke(DispID: Integer; const IID: TGUID; LocaleID: Integer; Flags: Word; var Params; VarResult, ExcepInfo, ArgErr: Pointer): HResult; stdcall;
end;
```

При обращении к методу интерфейса компонента в методе позднего связывания выполняются следующие действия:

1) имя вызываемой процедуры передается в метод GetIDsOfNames, которая возвращает числовой идентификатор, однозначно соответствующий имени метода сервера;
2) числовой идентификатор передается в Invoke для получения адреса на вызываемый метод интерфейса.

Промежуточным звеном между интерфейсами и переменными типа Variant являются *интерфейсы диспетчеризации* (*dispinterface*). Интерфейсы диспетчеризации описываются почти так же, как и обычные интерфейсы:

```
IMyInterfase=interface(IDispatch)
    [<IID интерфейса>]
    {<метод интерфейса>; safecall;}
end;

IMyInterfaseDisp=dispinterface
    [<IID интерфейса>]
    {<метод интерфейса>; dispid <числовой идентификатор метода>;}
end;
```

IID интерфейса IMyInterface и IMyInterfaceDisp обязательно должны быть одинаковыми. Методы у IMyInterfaceDisp являются отображением методов IMyInterface, но вместо safecall используется dispid для того, чтобы задать методу собственный числовой идентификатор.

Все методы сервера автоматизации должны возвращать значение типа HResult. Возвращаемое значение является индикатором, нормально ли завершил метод запрограммированную в нем операцию, или выполнить ее не удалось. Другие данные, полученные в результате работы функции, возвращаются через параметры с модификатором out. Директива safecall инструктирует компилятор Delphi автоматически создать вокруг всех методов своего рода оболочку в виде конструкции try-except. Данные соглашения в Delphi определены по умолчанию, поэтому при обращении или реализации компонента автоматизации методы его интерфейсов выглядят как обычные методы. Например:

```
//Так должен выглядеть метод инт-са, производного от IDspatch
function MyMethod(out param: <тип переменной>): HResult; stdcall;

//Так метод инт-са, производного от IDspatch, реализован в Delphi
function MyMethod: <тип переменной>;
```

В интерфейсах, поддерживающих IDispatch, можно задавать не только *методы*, но и *свойства*. Свойства похожи на переменные, но для ввода и вывода данных в свойствах используются функции, названия которых обычно начинаются с префиксов Get_ (для считывания) и Set_ (для записи) соответственно. Свойства могут быть доступны не только на чтение и запись одновременно, но и только на чтение или только на запись. Процедуры Get_ и Set_, соответствующие свойству, остаются доступными для тех случаев, когда в каком-либо языке программирования реализация свойств не поддерживается.

Применение интерфейсов диспетчеризации обеспечивает быстрый вызов методов интерфейса при позднем связывании, так как в этом случае системе не требуется вызывать процедуру GetIDsOfNames. В компоненте реализуется только обычный интерфейс, а интерфейс диспетчеризации может применяться только в процессе обращения клиента к COM-серверу.

5.3. Пример реализации встроенного COM-сервера автоматизации

В качестве примера реализации встроенного COM-сервера автоматизации реализуем компонент перевода целого числа в двоичный, восьмеричный и шестнадцатеричный вид, который будет выводиться в виде строки.

1. Выполнив действия, описанные в п. 2.2, сохраним проект под именем Convert.dpr.
2. Затем выберем элемент Automation Object в диалоговом окне New Items (п. 2.3). В поле Class Name, мастера Automation Object Wizard введем ObjConvert. Нажав на кнопку OK, получим модуль следующего содержания:

```
unit Unit1;

interface

uses
    ComObj, ActiveX, ObjConvert_TLB, StdVcl;
type
    TObjConvert_ = class(TAutoObject, IObjConvert)
    protected
        { Protected declarations }
    end;

implementation

uses ComServ;

initialization
    TAutoObjectFactory.Create(ComServer, TObjConvert_,
                Class_ObjConvert_, ciMultiInstance, tmApartment);
end.
```

3. Аналогично, как и в примере п. 3.4, опишем в интерфейсе IObjConvert методы Bin, Hex, Oct.

4. Возвратимся в модуль Unit1. В разделе описания типов дано начальное описание нашего компонента. Исходя из этого описания, наш компонент наследуется от класса TAutoComObject и поддерживает интерфейс IObjConvert. В разделе protected объявлены три функции:

```
TObjConvert = class(TComObject, IConvert)
protected
    {Declare IConvert methods here}
    function Bin (const n: Word; var Str: WideString): Boolean;
    stdcall;
    function Oct(const n: Word; var Str: WideString): Boolean;
    stdcall;
    function Hex(const n: Word; var Str: WideString): Boolean;
    stdcall;
end;
```

5. Остается только реализовать функции Bin, Hex и Oct аналогично, как и в примере п. 3.4.
6. Откомпилировав приложение, получим COM-сервер, готовый к использованию. Он будет находиться в библиотеке Convert.dll.

5.4. Пример реализации внешнего COM-сервера автоматизации

Внешние COM-серверы обычно целесообразно создавать только в тех случаях, когда сервер должен работать как автономное приложение со своими собственными правами.

Примечание 5.2. *Удаленный доступ, т. е. доступ от клиента, выполняемого на другом компьютере с использованием DCOM, возможен только к внешнему серверу автоматизации.*

Приведем простой пример реализации внешнего COM-сервера, который поддерживает компонент автоматизации для изменения цвета формы клиентом.

1. Создадим новый проект-приложение MyServer.dpr со своей формой.
2. Поместим в проект объект автоматизации, выбрав на закладке ActiveX диалогового окна New Items (рис. 2.1) элемент Automation Object. В поле Class Name мастера Automation Object Wizard укажем AutoServer. Нажав на кнопку OK, получим шаблон для реализации компонента автоматизации.
3. Используя редактор библиотеки типов, зададим свойство Color, доступное для чтения и записи, в интерфейсе IObjServer.

4. Реализуем методы Get_Color и Set_Color:

```
function TAutoServer.Get_Color: SYSUINT;

begin
    result:=Form1.Color;
end;

procedure TAutoServer.Set_Color(Value: SYSUINT);

begin
    Form1.Color:=Value;
end;
```

5. Откомпилируйте и зарегистрируйте полученное приложение.

Чтобы увидеть работу клиента и внешнего COM-сервера, реализуем для нашего примера клиент.

1. Создайте новый проект-приложение MyClient.dpr со своей формой.
2. В разделе описания private формы создаваемого приложения введите переменную Server типа Variant.
3. На событие OnCreate и OnDestroy формы установите процедуру вызова нашего сервера:

```
procedure TForm1.FormShow(Sender: TObject);

begin
    Server:=CreateOLEObject('MyServer.AutoServer');
end;

procedure TForm1.FormDestroy(Sender: TObject);

begin
    Server:=NULL;
end;
```

4. На форме установите ColorDialog и Button1. Реализуйте событие OnClick кнопки:

```
procedure TForm1.Button1Click(Sender: TObject);

begin
    if ColorDialog1.Execute then Server.Color:=ColorDialog1.Color;
end;
```

5. Откомпилируйте полученное приложение.

При старте приложения-клиента автоматически запускается внешний COM-сервер. Форма сервера меняет свой цвет в соответствии с выбранным цветом в диалоговом окне клиента, вызываемом при нажатии на кнопку Button1. Обратите внимание, когда клиент завершает работу, сервер тоже выгружается.

5.5. События в COM и обратные вызовы

При разработке серьезных COM-приложений часто требуется выполнять программный код клиента, если в COM-сервере произошло то или иное действие. Особенно это актуально, если COM-сервер поддерживает диалоговые окна для ввода каких-либо данных и передачи результата клиенту. Основная идея механизма событий следующая: клиент при подключении к COM-серверу передает интерфейс, в котором реализованы функции событий. Интерфейс, передаваемый клиентом COM-серверу, называется *интерфейсом обратного вызова*.

Для реализации компонента, поддерживающего механизм обработки событий и обратного вызова, в мастере Automation Object Wizard нужно установить флажок Generate Event support code. Это позволит сгенерировать шаблон создаваемого компонента с поддержкой событий:

```
<имя компонента> = class(TAutoObject, IConnectionPointContainer,
                                        {<интерфейсы компонента>})
private
    { Private declarations }
  FConnectionPoints: TConnectionPoints;
  FConnectionPoint: TConnectionPoint;
  FSinkList: TList;
  FEvents: <интерфейс событий>;
public
  procedure Initialize; override;
protected
    { Protected declarations }
  property ConnectionPoints: TConnectionPoints
      read FConnectionPoints implements IConnectionPointContainer;

  procedure EventSinkChanged(const EventSink: IUnknown); override;
end;
```

Это основные свойства и методы для поддержки компонентом COM-сервера механизма событий и функции обратного вызова. Процедуры

Intialize и EventSinkChanged автоматически реализуются средой программирования Delphi:

```
procedure <имя компонента>.EventSinkChanged(const
                                        EventSink: IUnknown);
begin
    FEvents := EventSink as <интерфейс событий>;
    if FConnectionPoint <> nil then
        FSinkList := FConnectionPoint.SinkList;
end;

procedure <имя компонента>.Initialize;
begin
    inherited Initialize;
    FConnectionPoints := TConnectionPoints.Create(Self);
    if AutoFactory.EventTypeInfo <> nil then
        FConnectionPoint := FConnectionPoints.CreateConnectionPoint(
                    AutoFactory.EventIID, ckSingle, EventConnect)
    else FConnectionPoint := nil;
end;
```

Когда создается компонент автоматизации с поддержкой событий, в библиотеке типов вводятся два интерфейса: интерфейс компонента и интерфейс диспетчеризации событий (обычно у него окончание Events).

Примечание 5.3. *Интерфейсом событий может быть любой интерфейс диспетчеризации, но тогда придется вручную его установить в Со-класс (закладка implements в библиотеке типов) и задать ему значения Source и Default (основной интерфейс будет поддерживать Default).*

FConnectionPoints – агрегированный объект, в котором реализован интерфейс IConnectionPointContainer, который необходим для поддержки механизма событий.

FConnectionPoint – элемент объекта FConnectionPoints.

FSinkList – список интерфейсов клиентов.

FEvents – интерфейс обратного вызова. Если нужно вызвать функцию, отвечающую за какое-либо событие, то необходимо написать следующий код:

```
FEvents.< функция события>;
```

Примечание 5.4. *Перед тем, как вызвать функцию события, используйте проверку наличия ссылки в переменной FEvents:*

if *Assigned(FEvents)* ***then*** *FEvents.< функция события>;*

Это необходимо для избежания ошибок, когда компонент не получил от клиента интерфейс обратного вызова.

В Delphi автоматически создается программный код, поддерживающий механизм обработки событий только для первого в порядке подключений клиента. Чтобы была возможность передавать события множеству подключенных клиентов, нужно внести некоторые изменения в программный код.

Во-первых, для возможности получения ссылки на данный COM-сервер с помощью функции GetOleObject его нужно зарегистрировать в таблице выполняемых объектов Windows. Для этих целей в класс проектируемого компонента нужно добавить поле FObjectID: integer для хранения идентификатора, а в процедуру Intitialize добавить процедуру регистрации компонента:

```
RegisterActiveObject(self as Iunknown, <CLSID компонента>, AC-
                                TIVEOBJECT_WEAK, FObjectID)
```

Для того чтобы удалить компонент из таблицы выполняемых объектов Windows, нужно создать метод Destroy и в нем вызвать функцию RevokeActiveObject:

```
destructor <имя компонента>.Destroy;

begin
    RevokeActiveObject(FObjectID, nil);
end;
```

При создании элемента FConnectionPoint с помощью метода CreateConnectionPoint в качестве предпоследнего параметра передается не ckSingle (заданный по умолчанию), а ckMulti. В итоге реализация метода Initialize будет следующая:

```
procedure <имя компонента>.Initialize;
begin
    inherited Initialize;
    FConnectionPoints := TConnectionPoints.Create(Self);
    if AutoFactory.EventTypeInfo <> nil then
        FConnectionPoint := FConnectionPoints.CreateConnectionPoint
                    (AutoFactory.EventIID, ckMulti, EventConnect)
    else FConnectionPoint := nil;
    RegisterActiveObject(self as Iunknown, <CLSID компонента>, AC-
                                TIVEOBJECT_WEAK, FObjectID)
end;
```

Для того чтобы отслеживать множество параллельных подключений, нужно реализовать следующую функцию:

```
function <имя компонента>.GetEnumerator: IEnumConnections;

var
```

```
    Container: IConnectionPointContainer;
    ConnectionPoint: IconnectionPoint;
begin
    OleCheck(QueryInterface(IConnectionPointContainer, Container));
    OleCheck(Container.FindConnectionPoint(AutoFactory.EventIID,
                ConnectionPoint));
    ConnectionPoint.EnumConnections(result);
end;
```

Данная функция позволяет получить данные от счетчика подключений, после чего нужно «пройтись» по всем подключениям:

```
var
    Enum: IEnumConnections;
    ConnectData: TConnectData;
    Fetched: Cardinal;
    <……….……>

begin {метод разрабатываемого компонента}
    <……….……>
    {обработка событий при подключении множества клиентов}
    Enum:=GetEnumerator;
    if Enum <> nil then
    begin
        while Enum.Next(1,ConnectData,@Fetched) = S_OK do
            if ConnectData.pUnk <> nil then
     (ConnectData.pUnk as <интерфейс событий>).<функция событий>
    end;
    <……….……>
end;
```

Рассмотрим далее реализацию приложений, использующих COM-серверы, поддерживающие события.

Примечание 5.5. *Реализация приложений-клиентов для COM-серверов, поддерживающих события, упрощена тем, что с помощью диалогового окна импорта библиотеки типов можно сгенерировать компонент Delphi, являющийся оболочкой, поддерживающей все свойства и методы компонента COM-сервера.*

Для того чтобы обрабатывать события подключенного компонента, необходимо создать в приложении объект, поддерживающий интерфейс, схожий с интерфейсом событий компонента. Отличие данных интерфейсов заключается в том, что в компоненте используется интерфейс диспетчеризации (dispinterface), а в приложении – обычный интерфейс.

Примечание 5.6. *Обратите внимание, что интерфейсы компонента и объекта в приложении должны иметь одинаковый IID. Также должна совпадать нумерация методов интерфейсов.*

При подключении приложения к компоненту нужно выполнить следующий программный код:

```
with <интерфейс компонента> as IConnectionPointContainer do
begin
    FindConnectionPoint(<IID интерфейса событий>, ESink);
    OleCheck(ESink.Advise(<компонент обработки событий> as
                        IDispatch, ConnectId));
end;
```

В данном программном коде выполняются следующие действия:

1. Ищется точка «входа» для соответствующего обработчика событий.
2. В компонент передается указатель на интерфейс объекта подключения для обработки событий. Функция ESink.Advise в параметре ConnectId возвращает идентификатор соединения, который будет использоваться для разрыва связи клиента с сервером.

При завершении работы с компонентом нужно разорвать связь между клиентом и сервером. Для этих целей используется следующий программный код:

```
with <интефейс компонента> as IConnectionPointContainer do
begin
    FindConnectionPoint(<IID интерфейса событий>, ESink);
    ESink.UnAdvise(ConnectId);
end;
```

Аналогично, как и при подключении, ищется точка «входа» для соответствующего обработчика событий, затем вызывается процедура разрыва соединения ESink.UnAdvise, в которую передается идентификатор соединения, полученный во время подключения к серверу.

Примечание 5.7. *Не забывайте отключаться от серверов, иначе система COM будет считать, что компонент COM-сервера еще используется и не сможет его выгрузить.*

VI. Автоматизация приложений Microsoft Office

Во многих приложениях, связанных с формированием документов и отчетов, нередко применяют сервисы, предоставляемые Microsoft Office. Программы, предоставляемые в данном пакете, разработаны по техноло-

гии COM, что обеспечивает возможность управления данным приложением с помощью других приложений и макроязыков.

Все компоненты Microsoft Office поддерживают один и тот же макроязык: Visual Basic for Applications (VBA), позволяющий создавать приложения непосредственно внутри документов Office (это называется «решениями на базе Microsoft Office»). Управление же компонентами Office из других приложений осуществляется с помощью *автоматизации* (Automation) – все приложения Microsoft Office являются *серверами автоматизации* (или COM-серверами). Для создания таких приложений пригодны любые средства разработки, позволяющие создавать контроллеры автоматизации (COM-клиенты). Наиболее часто для этой цели используется Visual Basic, но это могут быть и Delphi, C++ Builder, Visual C++ и т. п. Однако, прежде чем обсуждать возможности тех или иных средств разработки, следует разобраться, что такое автоматизация. Практически все, что в состоянии сделать пользователь любого приложения Microsoft Office с помощью меню, клавиатуры и инструментальной панели, может быть реализовано из приложения, созданного с помощью одного из средств разработки.

Основное преимущество применения приложений Microsoft Office связано с тем, что нет необходимости разрабатывать собственные средства формирования и просмотра отчетов. К тому же, некоторые пользователи, привыкшие использовать приложения Microsoft Office в повседневной работе, предпочитают хранить отчеты и другие документы в одном из форматов Microsoft Office.

В данной главе приведено описание методики реализации управления приложениями Word и Excel в программах, разрабатываемых в визуальной среде Delphi, на основе технологии автоматизации COM-приложений.

Современные визуальные среды Delphi (версии 7 и выше) позволяют создавать компоненты, внутри которых реализована связь с COM-серверами. С одной стороны, такой подход к программированию позволяет избежать некоторых трудностей, связанных с реализацией технологии COM, а с другой стороны – программы, реализованные на таких компонентах, работают существенно медленнее, чем те, которые используют интерфейсы напрямую. Другим отрицательным моментом является то, что некоторые методы имеют достаточно большое количество параметров (например, метод PrintOut приложения Word имеет около двадцати параметров), установленных по умолчанию, но в процессе формирования компонентов в среде Delphi данные значения теряются и их все равно приходится вводить.

Ввиду перечисленных неудобств большинство программистов предпочитают использовать *позднее связывание*, где реальная ссылка на объект в переменной или классе появляется на этапе выполнения приложения-контроллера. Хотя этот способ более медленный, чем ранее связывание, но он доступен во всех средствах разработки, позволяющих создавать контроллеры автоматизации, и менее чувствителен к тому, все ли параметры методов перечислены при их вызовах в коде приложения-контроллера. При позднем связывании корректность вызовов методов проверяется в момент их осуществления, то есть на этапе выполнения приложения, а не на этапе его компиляции.

Во всех примерах данной главы используется раннее связывание.

6.1. VBA и средства разработки контроллеров автоматизации

Как узнать, какие объекты доступны в серверах автоматизации приложений Microsoft Office? Для этой цели можно использовать документацию, описывающую их объектную модель [3]. Также могут быть полезными справочные файлы для программистов на Visual Basic for Applications.

Иногда для освоения объектной модели автоматизируемого сервера можно начать с записи необходимой последовательности действий в виде макроса с помощью VBA. Создать макрос можно, выбрав из меню приложения Microsoft Office пункт Tools/Macro/Record New Macro. Просмотр полученного макроса в редакторе кода VBA (рис. 6.1) обычно позволяет понять, как должен выглядеть код, реализующий эту последовательность действий.

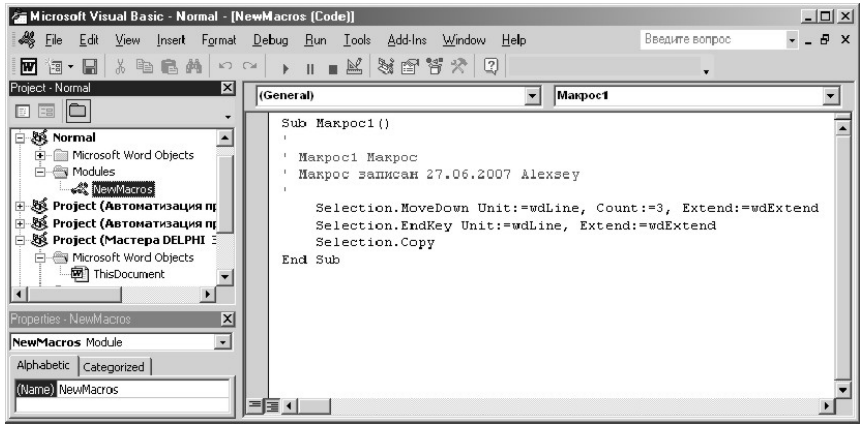

Рис. 6.1. Редактор кода VBA

Таким образом, чтобы узнать, как реализовать какие-либо действия, используя объекты автоматизации, необходимо выполнить следующие действия:

1) записать последовательность действий в макрос;

2) рассмотреть полученный код в редакторе (рис. 1) и перевести его на язык программирования, который используется для разработки приложения;

3) для поиска дополнительной информации можно использовать информационно-справочную систему MSDN [1] или VBAxxx9.CHM (для Microsoft Office 97 — VBxxx8.HLP соответственно).

Примечание 6.1. *Параметры некоторых функций серверов автоматизации задаются в виде констант, которые не установлены в Delphi. Чтобы получить значение таких констант в VisualBasic, надо внутри макроса написать MsgBox(<нужная_константа>).*

6.2. Объектные модели Microsoft Office

В объектных моделях всех приложений Microsoft Office всегда имеется самый главный объект, доступный приложению-контроллеру и представляющий само автоматизируемое приложение. Для всех приложений Microsoft Office он называется Application, и многие его свойства и методы для всех этих приложений также одинаковы. Наиболее часто используются следующие из них.

1. Свойство Visible (доступное для объекта Application всех приложений Microsoft Office) позволяет приложению появиться на экране и в панели задач; оно принимает значения True (пользовательский интерфейс приложения доступен) или False (пользовательский интерфейс приложения недоступен; это значение устанавливается по умолчанию). Если вам нужно сделать что-то с документом Office в фоновом режиме, не информируя об этом пользователя, можно не обращаться к этому свойству — в этом случае приложение можно будет найти только в списке процессов с помощью приложения Task Manager.

2. Метод Quit закрывает приложение Office. В зависимости от того, какое приложение Office автоматизируется, оно может иметь параметры или не иметь таковых.

Примечание 6.2. *В некоторых программах метод Quit может иметь параметры (например, в Microsoft Word), однако при позднем связывании их можно не учитывать.*

6.3. Общие принципы создания контроллеров автоматизации

Приложение, в котором используются серверы автоматизации, далее будут называться *контроллерами автоматизации*. В общем случае контроллер автоматизации должен выполнять следующие действия:

1) проверить, запущена ли копия приложения-сервера;

2) в зависимости от результатов проверки запустить копию автоматизируемого приложения Office либо подключиться к уже имеющейся копии;

3) сделать окно приложения-сервера видимым (в общем случае это не обязательно);

4) выполнить какие-то действия с приложением-сервером (например, создать или открыть документы, изменить их данные, сохранить документы и пр.);

5) закрыть приложение-сервер, если его копия была запущена данным контроллером, или отключиться от него, если контроллер подключился к уже имеющейся копии.

Для этих целей можно воспользоваться функциями GetActiveOleObject для подключения к уже запущенной копии приложения-сервера и CreateOleObject для запуска новой, если сервер не запущен. Например, чтобы создать новый экземпляр COM-сервера приложения Word, необходимо использовать следующий код:

App := CreateOleObject('App.Application');

В результате в переменную вариантного типа App будет передан основной интерфейс приложения.

6.4. Автоматизация Microsoft Word

В данном разделе мы обсудим наиболее часто встречающиеся задачи, связанные с автоматизацией Microsoft Word. Но перед этим рассмотрим, каковы программные идентификаторы основных объектов Microsoft Word и что представляет собой его объектная модель.

6.4.1. Программные идентификаторы и объектная модель Microsoft Word

Для приложения-контроллера доступны непосредственно следующие объекты:

Объект	Программный идентификатор	Комментарий
Application	Word.Application, Word.Application.9	создается экземпляр Word без открытых документов
Document	Word.Document, Word.Document.9, Word.Template.8	создается экземпляр Word с одним вновь созданным документом

Все остальные объекты Word являются так называемыми внутренними объектами. Это означает, что они не могут быть созданы сами по себе; так, объект Paragraph (абзац) не может быть создан отдельно от содержащего его документа.

Если вспомнить, что основное назначение Word — работа с документами, можно легко понять иерархию его объектной модели (рис. 2).

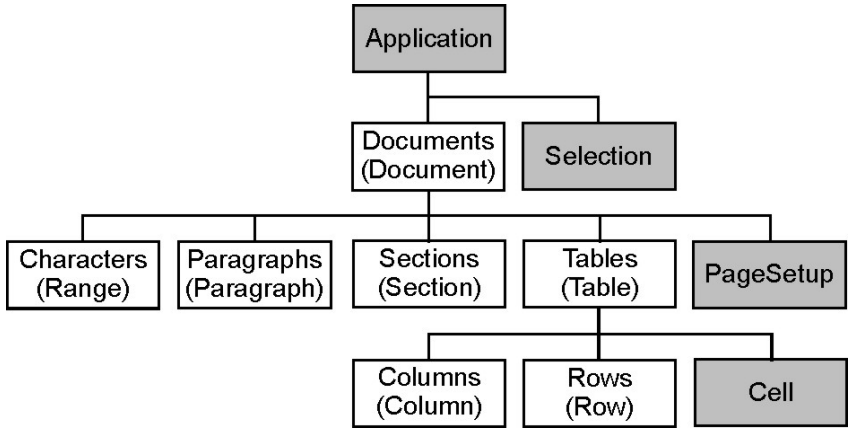

Рис. 6.2. Фрагмент объектной модели Microsoft Word

Основным объектом в ней, как и во всех других приложениях Microsoft Office, является объект Application, содержащий коллекцию Documents объектов типа Document. Каждый объект типа Document содержит коллекцию Paragraphs объектов типа Paragraph, Bookmarks типа Bookmark, Characters типа Character и т.д. Манипуляция документами, абзацами, символами, закладками реально осуществляется путем обращения к свойствам и методам этих объектов.

6.4.2. Создание и открытие документов Microsoft Word

Для создания примеров использования Microsoft Word можно использовать код создания контроллера, приведенный выше, и модифицировать его, заменяя комментарии кодом, манипулирующим свойствами и методами объекта Word.Application. Мы начнем с создания и открытия документов.

Создать новый документ Word можно, используя метод Add коллекции Documents объекта Application:

```
App.Documents.Add;
```

Чтобы создать нестандартный документ, нужно указать имя шаблона в качестве параметра метода Add:

```
App.Documents.Add('C:\Program Files\ Microsoft Office\Templates\1033\Manual.dot');
```

Для открытия уже существующего документа следует воспользоваться методом Open коллекции Documents:

```
App.Documents.Open('C:\MyWordFile.doc');
```

Отметим, что свойство ActiveDocument объекта Word.Application указывает на текущий активный документ среди одного или нескольких открытых. Помимо этого к документу можно обращаться по его порядковому номеру с помощью метода Item; например ко второму открытому документу можно обратиться так:

```
App.Documents.Item(2)
```

Примечание 6.3. *Нумерация членов коллекций в Microsoft Office начинается с единицы.*

Сделать документ активным можно с помощью метода Activate:

```
App.Documents.Item(1).Activate;
```

Следующее, чему следует научиться, – это сохранять документ Word и закрывать сам Word.

6.4.3. Сохранение, печать и закрытие документов Microsoft Word

Закрытие документа может быть осуществлено с помощью метода Close:

```
App.Documents.Item(2).Close;
```

или

```
App.ActiveDocument.Close;
```

Метод Close имеет несколько необязательных (в случае позднего связывания) параметров, влияющих на правила сохранения документа. Первый из них влияет на то, сохраняются ли внесенные в документ изменения, и принимает три возможных значения (соответствующие константы рекомендуется описать в приложении). Третий параметр принимает значения True или False и влияет на то, пересылать ли документ следующему пользователю по электронной почте. Если эта функциональность не применяется, можно проигнорировать этот параметр. Таким образом, при использовании этих параметров закрыть документ можно, например, так:

App.ActiveDocument.Close(wdSaveChanges, wdPromptUser) ;

Просто сохранить документ, не закрывая его, можно с помощью метода Save:

App.ActiveDocument.Save;

Этот метод также имеет несколько необязательных (в случае позднего связывания) параметров, первый из которых равен True, если документ сохраняется автоматически, и False, если нужно выводить диалоговую панель для получения подтверждения пользователя о сохранении изменений (если таковые были сделаны). Второй параметр влияет на формат сохраняемого документа, и список его возможных значений совпадает со списком значений второго параметра метода Close.

Напоминаем, что закрыть сам Word можно с помощью метода Quit объекта Word.Application. Этот метод имеет в общем случае три параметра, совпадающих с параметрами метода Close объекта Document.

Вывод документа на устройство печати можно осуществить с помощью метода PrintOut объекта Document, например:

App.ActiveDocument.PrintOut;

Если нужно изменить параметры печати, следует указать значения соответствующих параметров метода PrintOut (в случае Microsoft Word их около двадцати).

6.4.4. Вставка текста и объектов в документ и форматирование текста

Для создания абзацев в документе можно использовать коллекцию Paragraphs объекта Document, представляющую набор абзацев данного документа. Добавить новый абзац можно с помощью метода Add этой коллекции:

App.ActiveDocument.Paragraphs.Add;

Для вставки собственно текста в документ, тем не менее, применяется не объект Paragraph, а объект Range, представляющий любую непрерывную часть документа (в том числе и вновь созданный абзац). Этот объект может быть создан разными способами. Например, можно указать

начальный и конечный символы диапазона (если таковые имеются в документе):

```
var
  Rng : Variant;
...
  Rng := App.ActiveDocument.Range(2,4);
//со 2-го по 4-й символы
```

или указать номер абзаца (например, только что созданного):

```
Rng:= App.ActiveDocument.Paragraphs.Item(1).Range;
```

или указать несколько абзацев, следующих подряд:

```
Rng := App.ActiveDocument.Range
(App.ActiveDocument.Paragraphs.Item(3).Range.Start,
App.ActiveDocument.Paragraphs.Item(5).Range.End)
```

Вставить текст можно с помощью методов объекта Range InsertBefore (перед диапазоном) или InsertAfter (после диапазона), например:

```
Rng.InsertAfter('Это вставляемый текст');
```

Помимо объекта Range текст можно вставлять с помощью объекта Selection, являющегося свойством объекта Word.Application и представляющего собой выделенную часть документа (этот объект создается, если пользователь выделяет часть документа с помощью мыши, и может быть также создан с помощью приложения-контроллера). Сам объект Selection можно создать, применив метод Select к объекту Range, например:

```
var
  Sel : Variant;
...
App.ActiveDocument.Paragraphs.Item(3).Range.Select;
```

В приведенном выше примере в текущем документе выделяется третий абзац.

Если мы хотим вставить строку текста в документ либо вместо выделенного фрагмента текста, либо перед ним, это можно сделать с помощью следующего фрагмента кода:

```
var
  Sel : Variant;
...
  Sel := App.Selection;
  Sel.TypeText('Это текст, которым мы заменим выделенный фрагмент');
```

Отметим, что если свойство Options.ReplaceSelection объекта Word.Application равно True, выделенный текст будет заменен на новый

текст (этот режим действует по умолчанию); если же нужно, чтобы текст был вставлен перед выделенным фрагментом, а не вместо него, следует установить это свойство равным False:

```
App.Options.ReplaceSelection := False;
```

Символ конца абзаца при использовании объекта Selection может быть вставлен с помощью следующего фрагмента кода:

```
Sel.TypeParagraph;
```

К объекту Selection, так же как и к объекту Range, можно применить методы InsertBefore и InsertAfter. В этом случае, в отличие от предыдущего, вставляемый текст станет частью выделенного фрагмента текста.

С помощью объекта Selection, используя его свойство Font и свойства объекта Font, такие как Bold, Italic, Size, …, можно отформатировать текст. Например, таким образом можно вставить строку, выделенную жирным шрифтом:

```
Sel.Font.Bold := True;
Sel.TypeText('Это текст, который мы выделим жирным шрифтом.');
Sel.Font.Bold := False;
Sel.TypeParagraph;
```

Для наложения на вставляемый текст определенного заранее стиля можно использовать свойство Style этого же объекта, например:

```
Sel.Style := 'Heading 1';
Sel.TypeText('Это текст, который станет заголовком');
Sel.TypeParagraph;
```

Нередко документы Word содержат данные других приложений. Простейший способ вставить такие данные в документ – использовать метод Paste объекта Range:

```
Var
  Rng: Variant;
...
  Rng := App.Selection.Range;
  Rng.Collapse(wdCollapseEnd);
  Rng.Paste;
```

Естественно, в этом случае в буфере обмена уже должны содержаться вставляемые данные.

Если нужно поместить в буфер обмена часть документа Word, это можно сделать с помощью метода Copy объекта Range:

```
Var
  Rng: Variant;
...
```



```
  Rng := App.Selection.Range;
  Rng.Copy;
```

Следующий этап – перемещение курсора в нужное место текста.

6.4.5. Перемещение курсора по тексту

Используя метод Collapse, можно «сжать» объект Range или объект Selection, сократив его размер до нуля символов:

```
Rng.Collapse(wdCollapseEnd);
```

Параметр этого метода указывает, в начале или в конце исходного фрагмента окажется новый объект Range или Selection. Если используется позднее связывание, то нужно определить в приложении соответствующие константы:

```
const
  wdCollapseStart = $00000001;//новый объект находится в начале фрагмента
  wdCollapseEnd = $00000000;//новый объект находится в конце фрагмента
```

Перемещать курсор по тексту можно с помощью метода Move объектов Range и Selection. Этот метод имеет два параметра. Первый указывает на то, в каких единицах измеряется перемещение — в символах (по умолчанию), словах, предложениях, абзацах и др. Второй параметр указывает, на сколько единиц при этом нужно переместиться (это число может быть и отрицательным; по умолчанию оно равно 1). Например, следующий фрагмент кода:

```
Rng.Move;
```

приведет к перемещению курсора на один символ вперед, а

```
Rng.Move(wdParagraph,3);
```

приведет к перемещению курсора на три абзаца вперед. Отметим, что этот метод использует следующие константы:

```
const        //Единицей перемещения является:
  wdCharacter = $00000001; //символ
  wdWord = $00000002; //слово
  wdSentence = $00000003; //предложение
  wdParagraph = $00000004; //абзац
  wdStory = $00000006; //часть документа (напр., колонтитул, оглавление и др.)
  wdSection = $00000008; //раздел
  wdColumn = $00000009; //колонка таблицы
  wdRow = $0000000A; //строка таблицы
  wdCell = $0000000C; //ячейка таблицы
  wdTable = $0000000F; //таблица
```

Нередко для перемещения по тексту используются закладки. Создать закладку в текущей позиции курсора можно путем добавления члена коллекции Bookmarks объекта Document с помощью метода Add, указав имя закладки в качестве параметра, например:

App.ActiveDocument.Bookmarks.Add('MyBookmark');

Проверить существование закладки в документе можно с помощью метода Exists, а переместиться на нее – с помощью метода Goto объектов Document, Range или Selection:

Rng := App.ActiveDocument.Goto(wdGoToBookmark, wdGoToNext, ,'MyBookmark');

Rng.InsertAfter('Текст, вставленный после закладки');

Значения констант для этого примера таковы:

```
const
wdGoToBookmark = $FFFFFFFF; //перейти к закладке
wdGoToNext = $00000002; //искать следующий объект в тексте
```

Отметим, что с помощью метода Goto можно перемещаться не только на указанную закладку, но и на другие объекты (рисунки, грамматические ошибки и др.), и направление перемещения тоже может быть различным. Поэтому список констант, которые могут быть использованы в качестве параметров данного метода, довольно велик.

6.4.6. Создание таблиц

Создавать таблицы можно двумя способами. Первый заключается в вызове метода Add коллекции Tables объекта Document и последовательном заполнении ячеек данными. Этот способ при позднем связывании работает довольно медленно.

Второй способ, намного более «быстрый», заключается в создании текста из нескольких строк, содержащих подстроки с разделителями (в качестве разделителя можно использовать любой или почти любой символ, нужно только, чтобы он заведомо не встречался в данных, которые будут помещены в будущую таблицу), и последующей конвертации такого текста в таблицу с помощью метода ConvertToTable объекта Range. Ниже приведен пример создания таблицы из трех строк и трех столбцов этим способом (в качестве разделителя, являющегося первым параметром метода ConvertToTable, используется запятая):

```
Var
  Rng: Variant;
...
  Rng := App.Selection.Range;
  Rng.Collapse(wdCollapseEnd);
  Rng.InsertAfter('1, 2, 3');
  Rng.InsertParagraphAfter;
```



```
  Rng.InsertAfter('4,5,6');
  Rng.InsertParagraphAfter;
  Rng.InsertAfter('7,8,9');
  Rng.InsertParagraphAfter;
  Rng.ConvertToTable(',');
```

Внешний вид таблицы можно изменить с помощью свойства Format, а также с помощью свойств коллекции Columns, представляющей колонки таблицы, и коллекции Rows, представляющей строки таблицы объекта Table.

6.4.7. Обращение к свойствам документа

Свойства документа можно получить с помощью коллекции BuiltInDocumentProperties объекта Document, например:

```
Memo1.Lines.Add('Название -
'+App.ActiveDocument.BuiltInDocumentProperties [wdPropertyTitle].Value);
Memo1.Lines.Add('Автор -
'+App.ActiveDocument.BuiltInDocumentProperties [wdPropertyAu-
thor].Value);
Memo1.Lines.Add('Шаблон -
'+App.ActiveDocument.BuiltInDocumentProperties [wdPropertyTem-
plate].Value);
```

В данном разделе приведены основные операции, которые наиболее часто применяются при автоматизации Microsoft Word. Естественно, возможности автоматизации Word далеко не исчерпываются приведенными примерами, однако, руководствуясь основными принципами создания контроллеров Word, изложенными в данной главе, и соответствующей справочной информацией [**Ошибка! Источник ссылки не найден.**], можно ими воспользоваться самостоятельно.

6.5. Автоматизация Microsoft Excel

В данном разделе рассматриваются наиболее часто встречающиеся задачи, связанные с автоматизацией Microsoft Excel. Но перед этим рассмотрим, каковы программные идентификаторы основных объектов Microsoft Excel и что представляет собой его объектная модель.

6.5.1. Программные идентификаторы и объектная модель Microsoft Excel

Существует три типа объектов Excel, которые могут быть созданы непосредственно с помощью приложения-контроллера. Эти объекты и соответствующие им программные идентификаторы перечислены ниже.

Объект	Программный идентификатор	Комментарий
Application	Excel.Application, Excel.Application.9	С помощью этого программного идентификатора создается экземпляр приложения без открытых рабочих книг
WorkBook	Excel.AddIn	С помощью этого программного идентификатора создается экземпляр расширения (add-in) Excel (имеющиеся расширения доступны с помощью пункта меню Tools/Add-Ins)
Chart	Excel.Chart, Excel.Chart.8	Рабочая книга, созданная с помощью этих программных идентификаторов, состоит из двух листов – одного для диаграммы, другого – для данных, на основе которых она построена
WorkSheet	Excel.Sheet, Excel.Sheet.8	Рабочая книга, созданная с помощью этих программных идентификаторов, состоит из одного листа

Все остальные объекты Excel являются внутренними объектами и существовать самостоятельно не могут.

Основным в объектной модели Excel является объект Application, содержащий коллекцию Workbooks объектов типа WorkBook (рис. 3).

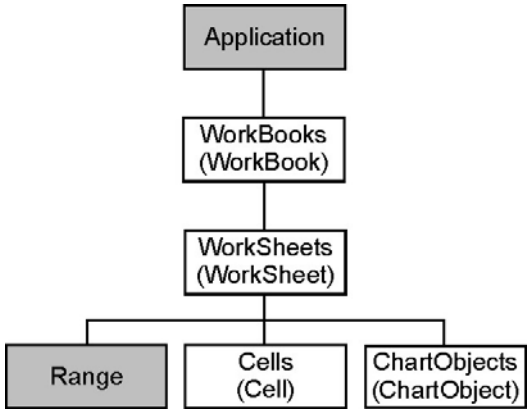

Рис. 6.3. Фрагмент объектной модели Microsoft Excel

Каждый объект типа WorkBook содержит коллекцию WorkSheets-объектов типа WorkSheet, Charts типа Chart и др. Манипуляция рабочими книгами, их листами, ячейками, диаграммами реально осуществляется путем обращения к свойствам и методам этих объектов.

Рассмотрим наиболее часто встречающиеся задачи, связанные с автоматизацией Microsoft Excel.

6.5.2. Запуск Microsoft Excel, создание и открытие рабочих книг

Для создания примеров использования Microsoft Excel можно использовать функцию CreateOleObject:

```
App := CreateOleObject('Excel.Application');
```

Создать новую рабочую книгу Excel можно, используя метод Add коллекции Workbooks объекта Application:

```
App.WorkBooks.Add;
```

Для создания рабочей книги на основе шаблона следует указать его имя в качестве первого параметра метода Add:

```
App.WorkBooks.Add ('C:\Program Files\Microsoft Office\Templates\1033\-invoice.xlt');
```

В качестве первого параметра этого метода можно также использовать следующие константы:

```
const
  xlWBATChart = $FFFFEFF3; //рабочая книга состоит из листа с диаграммой
  xlWBATWorksheet = $FFFFEFB9; //рабочая книга состоит из листа с данными
```

В этом случае рабочая книга будет содержать один лист типа, заданного указанной константой (график, обычный лист с данными и др.)

Для открытия уже существующего документа следует воспользоваться методом Open коллекции WorkBooks:

```
App.Documents.Open('C:\MyExcelFile.xls');
```

Отметим, что свойство ActiveWorkBook объекта Excel.Application указывает на текущую активную рабочую книгу среди одной или нескольких открытых. Помимо этого к рабочей книге можно обращаться по ее порядковому номеру, например ко второй открытой рабочей книге можно обратиться так:

```
App.WorkBooks[2]
```

Обратите внимание на то, что в Delphi при использовании позднего связывания синтаксис, используемый для обращения к членам коллекций объектов Excel, отличен от синтаксиса, используемого для обращения к объектам Word – в случае Word мы использовали метод Item, а в случае Excel мы обращаемся к членам коллекции как к элементам массива. Если же вы используете Visual Basic, синтаксис, применяемый для обращения к членам коллекций, будет одинаков для всех коллекций Microsoft Office.

Сделать рабочую книгу активной можно с помощью метода Activate:

```
App.WorkBooks[2].Activate;
```

Рассмотрим далее сохранение рабочих книг в файлах.

6.5.3. Сохранение, печать и закрытие рабочих книг Microsoft Excel

Закрытие документа может быть осуществлено с помощью метода Close:

```
App.WorkBooks[2].Close;
```

или

```
App.ActiveWorkBook.Close;
```

Метод Close имеет несколько необязательных (в случае позднего связывания) параметров, влияющих на правила сохранения рабочей книги. Первый из параметров принимает значения True или False и влияет на то, сохранять ли изменения, внесенные в рабочую книгу. Второй параметр (типа Variant) — имя файла, в котором нужно сохранить рабочую книгу (если в нее были внесены изменения). Третий параметр, также принимающий значения True или False, влияет на то, следует ли пересылать документ следующему пользователю по электронной почте, и может быть проигнорирован, если эта функциональность не используется:

```
App.ActiveWorkBook.Close(True,'C:\MyWorkBook.xls');
```

Просто сохранить рабочую книгу, не закрывая ее, можно с помощью методов Save или SaveAs:

```
App.ActiveWorkBook.Save;
```

или

```
App.ActiveWorkBook.SaveAs('C:\MyWorkBook.xls');
```

Метод SaveAs имеет более десятка параметров, влияющих на то, как именно сохраняется документ (под каким именем, с паролем или без него, какова кодовая страница для содержащегося в ней текста и др.).

Закрыть сам Excel можно с помощью метода Quit объекта Excel.Application. В случае Excel этот метод параметров не имеет.

Вывод документа Excel на устройство печати можно осуществить с помощью метода PrintOut объекта WorkBook, например:

```
App.ActiveWorkBook.PrintOut;
```

Если нужно изменить параметры печати, следует указать значения соответствующих параметров метода PrintOut (в случае Excel их восемь).

6.5.4. Обращение к листам и ячейкам

Обращение к листам рабочей книги производится с помощью коллекции WorkSheets объекта WorkBook. Каждый член этой коллекции представляет собой объект WorkSheet. К члену этой коллекции можно обратиться по его порядковому номеру, например:

```
App.WorkBooks[1].WorkSheets[1].Name := 'Страница 1';
```

Приведенный выше пример иллюстрирует, как можно изменить имя листа рабочей книги.

К листу рабочей книги можно обратиться и по имени, например:

```
App.WorkBooks[1].WorkSheets['Sheet1'].Name := 'Страница 1';
```

Обращение к отдельным ячейкам листа производится с помощью коллекции Cells объекта WorkSheet. Например, добавить данные в ячейку B1 можно следующим образом:

```
App.WorkBooks[1].WorkSheets['Sheet1'].Cells[1,2].Value:='25';
```

Здесь первая из координат ячейки указывает на номер строки, вторая – на номер столбца.

Добавление формул в ячейки производится аналогичным способом:

```
App.WorkBooks[1].WorkSheets['Sheet1'].Cells[3,2].Value := '
=SUM(B1:B2)';
```

Очистить ячейку можно с помощью метода ClearContents.

Форматирование текста в ячейках производится с помощью свойств Font и Interior объекта Cell и их подсвойств. Например, следующий фрагмент кода выводит текст в ячейке красным жирным шрифтом Courier кегля 16 на желтом фоне:

```
App.WorkBooks[1].WorkSheets[1].Cells[3,2].Interior.Color := clYellow;
App.WorkBooks[1].WorkSheets[1].Cells[3,2].Font.Color := clRed;
App.WorkBooks[1].WorkSheets[1].Cells[3,2].Font.Name := 'Courier';
App.WorkBooks[1].WorkSheets[1].Cells[3,2].Font.Size := 16;
App.WorkBooks[1].WorkSheets[1].Cells[3,2].Font.Bold := True;
```

Вместо свойства Color можно использовать свойство ColorIndex, принимающее значения от 1 до 56; таблицу соответствий значений этого

свойства реальным цветам можно найти в справочном файле VBAXL9.CHM.

Обратиться к текущей ячейке можно с помощью свойства ActiveCell объекта Excel.Application, а узнать местоположение ячейки можно с помощью свойства Address объекта Cell, например:

```
ShowMessage(App.ActiveCell.Address);
```

Помимо обращения к отдельным ячейкам, можно манипулировать прямоугольными областями ячеек с помощью объекта Range, например:

```
App.WorkBooks[1].WorkSheets[2].Range['A1:C5'].Value := 'Test';
App.WorkBooks[1].WorkSheets[2].Range['A1:C5'].Font.Color := clRed;
```

Приведенный выше код приводит к заполнению прямоугольного участка текстом и к изменению цвета шрифта ячеек.

Объект Range также часто используется для копирования прямоугольных областей через буфер обмена. Ниже приведен пример, иллюстрирующий копирование такой области:

```
App.WorkBooks[1].WorkSheets[2].Range['A1:C5'].Copy;
App.WorkBooks[1].WorkSheets[2].Range['A11:C15'].Select;
App.WorkBooks[1].WorkSheets[2].Paste;
```

Обратите внимание на то, что диапазон, куда копируются данные, предварительно выделяется с помощью метода Select.

Отметим, что примерно таким же образом можно копировать данные и из других приложений (например, из Microsoft Word).

Довольно часто при автоматизации Excel используются его возможности, связанные с построением диаграмм. Ниже мы рассмотрим, как это сделать.

6.5.5. Создание диаграмм

Диаграммам Excel соответствует объект Chart, который может располагаться как на отдельном листе, так и на листе с данными. Если объект Chart располагается на листе с данными, ему соответствует член коллекции ChartObjects объекта WorkSheet и создание диаграммы нужно начать с добавления элемента в эту коллекцию:

```
Ch:=App.WorkBooks[1].WorkSheets[2].ChartObjects.Add(10,50,400,400);
```

Параметрами этого метода являются координаты левого верхнего угла и размеры диаграммы в пунктах (1/72 дюйма).

Если же диаграмма располагается на отдельном листе (не предназначенном для хранения данных), то создание диаграммы нужно начать с добавления элемента в коллекцию Sheets объекта Application (которая отличается от коллекции WorkSheets тем, что содержит листы всех типов, а не только листы с данными):

```
App.WorkBooks[1].Sheets.Add(,,1,xlWBATChart);
```


В этом случае первый параметр метода Add указывает порядковый номер листа, перед которым нужно поместить данный лист (или данные листы, если их несколько), второй параметр – порядковый номер листа, после которого нужно поместить данный лист (используется обычно один из них), третий параметр – сколько нужно создать листов, а четвертый – какого типа должен быть лист. Значения четвертого параметра совпадают со значениями первого параметра метода Add коллекции WorkBooks объекта Application, и при использовании имен соответствующих констант следует определить их в приложении-контроллере.

Простейший способ создать диаграмму, с точки зрения пользователя, – создать ее с помощью соответствующего эксперта на основе прямоугольной области с данными. Точно так же можно создать диаграмму и с помощью контроллера автоматизации – для этой цели у объекта Chart, являющегося свойством объекта ChartObject (члена коллекции ChartObjects), имеется метод ChartWizard. Первым параметром этого метода является объект Range, содержащий диапазон ячеек для построения диаграммы, а вторым – числовой параметр, указывающий, какого типа должна быть эта диаграмма:

```
Var Ch: Variant;
…
Ch.Chart.ChartWizard(App.WorkBooks[1].WorkSheets[2].Range['A1:C5'],
xl3DColumn);
```

Возможные значения параметра, отвечающего за тип диаграммы, можно найти в справочном файле.

У объекта Chart имеется множество свойств, отвечающих за внешний вид диаграммы, с помощью которых можно изменить ее точно так же, как пользователи делают это вручную. Ниже приводится пример создания заголовка диаграммы и подписей вдоль ее осей (отметим, что оси есть не у всех типов диаграмм).

```
Ch.Chart.HasTitle :=1;
Ch.Chart.HasLegend := False;
Ch.Chart.ChartTitle.Text := 'Пример диаграммы Excel ';
Ch.Chart.Axes(1).HasTitle := True;
Ch.Chart.Axes(1).AxisTitle.Text := 'Подпись вдоль оси абсцисс';
Ch.Chart.Axes(2).HasTitle := True;
Ch.Chart.Axes(2).AxisTitle.Text := 'Подпись вдоль оси ординат';
```

Еще один способ создания диаграммы – определить все ее параметры с помощью свойств объекта Chart, включая и определение серий, на осно-

ве которых она должна быть построена. Данные для серии обычно находится в объекте Range, содержащем строку или столбец данных, а добавление серии к диаграмме производится путем добавления члена к коллекции SeriesCollection, например:

```
App.WorkBooks[1].Sheets.Add(,,1,xlWBATChart);
App.WorkBooks[1].Sheets[1].ChartType := xl3DPie;
Rng:=App.WorkBooks[1].WorkSheets[2].Range['B1:B5'];
App.WorkBooks[1].Sheets[1].SeriesCollection.Add(Rng);
```

В данном примере к диаграмме, созданной на отдельном листе, специально предназначенном для диаграмм, добавляется одна серия на основе диапазона ячеек другого листа.

Возможности автоматизации Microsoft Excel и Word далеко не исчерпываются приведенными примерами. Сведения о них можно всегда найти в соответствующем справочном файле.

6.6. Пример экспорта данных в Word

В качестве примера рассмотрим задачу формирования отчета данных из заданной базы данных в Microsoft Word: необходимо сформировать сложный документ с колонтитулами, таблицами и заголовками. Также документ должен иметь титульную страницу, на которой пишется соответствующий текст, со второй страницы начинается верхний колонтитул, где задается заголовок. Для подключения к базе данных будут использоваться стандартные компоненты Delphi, например, TADODataset.

Существует два способа решения поставленной задачи:

1. *Без использования шаблона*: создается новый документ, где создаются колонтитулы надписи и т.д. Этот метод достаточно сложен, так как необходимо учитывать то, что на разных компьютерах могут изначально стоять разные настройки страницы, шрифтов, абзацев.

2. *С использованием шаблона*: заранее создается документ, который используется как шаблон. Например, в шаблон в нужном месте в центре первой страницы ставится объект типа надпись, в которой выставляется нужный шрифт и выравнивание. В конце первой страницы делается разрыв раздела. На второй странице в верхний колонтитул тоже вставляется объект типа надпись. В него будет вставляться название документа.

Далее записывается макрос, в котором выполняются действия, которые необходимо реализовать в Delphi:

1) в объект типа надпись на первой странице внести текст;
2) перейти в конец документа;
3) перейти в верхний колонтитул;
4) в объект типа надпись в колонтитуле внести текст;
5) вернуться на вторую страницу.

В результате получается следующий макрос:

```
Sub Макрос1()
    ActiveDocument.Shapes("Text Box 8").Select
    Selection.TypeText Text:="текст на титульной"
    Selection.EndKey Unit:=wdStory
    If ActiveWindow.View.SplitSpecial <> wdPaneNone Then
        ActiveWindow.Panes(2).Close
    End If
    If ActiveWindow.ActivePane.View.Type = wdNormalView Or _
ActiveWindow.ActivePane.View.Type = wdOutlineView Then
        ActiveWindow.ActivePane.View.Type = wdPrintView
    End If
    ActiveWindow.ActivePane.View.SeekView = wdSeekCurrentPage-
Header
    Selection.HeaderFooter.Shapes("Text Box 5").Select
    Selection.TypeText Text:="Текст в колонтитуле"
    ActiveWindow.ActivePane.View.SeekView = wdSeekMainDocument
End Sub
```

Если преобразовать в код Delphi, то получим:

```
//выбирается первая надпись
Vr := 'Text Box 8';
Word.ActiveDocument.Shapes.Item(vr).Select;
//пишется текст
Word.Selection.TypeText('текст на титульной');
//переход в конец документа
Vr := wdStory;
Word.Selection.EndKey(Vr);
//операторы if здесь не нужны
//переход в верхний колонтитул
Word.ActiveWindow.ActivePane.View.SeekView := wdSeekCurrent-
PageHeader;
Vr := 'Text Box 5';
Word.Selection.HeaderFooter.Shapes.Item(Vr).Select;
Word.Selection.TypeText('текст в колонтитуле');
Word.ActiveWindow.ActivePane.View.SeekView := wdSeekMainDocu-
ment;
```

Аналогичным образом можно получить код ввода текста в колонтитулы, настройку шрифтов, абзацев и т. п.

Решение поставленной задачи приведено в следующем листинге:

```
unit Unit1;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls,
Forms,
  Dialogs, StdCtrls, DB, ADODB, ActiveX;

type
  TForm1 = class(TForm)
    ADOConnection1: TADOConnection;
    ADODataSet1: TADODataSet;
    Button1: TButton;
    procedure Button1Click(Sender: TObject);
  private
    { Private declarations }
    procedure TableExport(Word,Document,Selection: Variant; DataSet:
TDataSet; Title,FlagText: String);

  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

uses ComObj;

{$R *.dfm}

procedure TForm1.Button1Click(Sender: TObject);
var
  Word: Variant;
  Document: Variant;
  Selection: Variant;
  Vr: OleVariant;
begin
```



```
  Screen.Cursor := crHourGlass;
  //создается сервер автоматизации Word
  Word := CreateOleObject('Word.Application');
  try
    //создается новый документ по шаблону
    Document := Word.Documents.Add(GetCurrentDir+'\Shablon.doc');
    //выбирается первая надпись
    Vr := 'Text Box 8';
    Document.Shapes.Item(Vr).Select;
    //запоминается ссылка на объект Selection

    Selection := Word.Selection;
    //пишется текст
    Selection.TypeText('текст на титульной');
    //переход в конец документа
    Vr := wdStory;
    Selection.EndKey(Vr);
    //переход в верхний колонтитул
    Word.ActiveWindow.ActivePane.View.SeekView := wdSeekCurrent-
PageHeader;
    Vr := 'Text Box 5';
    Selection.HeaderFooter.Shapes.Item(Vr).Select;
    Word.Selection.TypeText('текст в колонтитуле');
    Word.ActiveWindow.ActivePane.View.SeekView := wdSeekMain-
Document;
    ADODataSet1.Close;
    ADODataSet1.CommandText := 'SELECT [Main].[Tel],
[Main].[name], '+
      '[Main].[House], [Korp].[NameKorp], [Flat].[NameFlat],
[Ul].[NameUl] '+
      ' FROM Main, Ul, Korp, Flat WHERE ([Main].[IdUl]=[Ul].[IdUl])
And '+
      '([Main].[IdKorp]=[Korp].[IdKorp]) And '+
      '([Main].[IdFlat]=[Flat].[IdFlat]) ';
    ADODataSet1.Open;
    //настраиваются параметры экспорта
    //заголовок

    ADODataSet1.fields[0].DisplayLabel := 'Телефон';
    //ширина столбца
    ADODataSet1.fields[0].Tag := round(Word.CentimetersToPoints(2));
    //отображать поле
    DataSet1.fields[0].Visible := True;
```

```
        //заголовок

        ADODataSet1.fields[1].DisplayLabel := 'ФИО';
        //ширина столбца

        ADODataSet1.fields[1].Tag := Round(Word.CentimetersToPoints(10));
       //отображать поле

        ADODataSet1.fields[1].Visible :=  True;
        //заголовок

        ADODataSet1.fields[2].DisplayLabel  :=  'Дом';
        //ширина столбца

        ADODataSet1.fields[2].Tag:= Round(Word.CentimetersToPoints(1.5));
        //отображать поле
        ADODataSet1.fields[2].Visible :=  True;
        //заголовок

        ADODataSet1.fields[3].DisplayLabel  :=  'Корпус';
        //ширина столбца

        ADODataSet1.fields[3].Tag :=
Round(Word.CentimetersToPoints(1.5));
        //отображать поле

        ADODataSet1.fields[3].Visible :=  True;
        //заголовок

        ADODataSet1.fields[4].DisplayLabel  :=  'Квартира';
        //ширина столбца

        ADODataSet1.fields[4].Tag :=
Round(Word.CentimetersToPoints(1.5));
        //отображать поле

        ADODataSet1.fields[4].Visible :=  True;
        //не отображать поле

        ADODataSet1.fields[5].Visible :=  False;
        //вызывается процедура экспорта
        TableExport(Word,Document,Selection,ADODataSet1,'Живущие на
улице Ахметова','АХМЕТОВА');
```



```
        TableExport(Word,Document,Selection,ADODataSet1,'Живущие на
улице Летчиков','ЛЕТЧИКОВ');
      finally
        //отображется Word
        Word.Visible := True;
        //не забудьте удалить ссылки!!!
        Selection := Null;
        Document := Null;
        Word := Null;
        Screen.Cursor := crDefault;
      end;
    end;

    procedure TForm1.TableExport(Word,Document,Selection: Variant;
DataSet: TDataSet; Title,FlagText: String);
    var
      I,ColCount: Integer; //количество колонок в таблице
      TableBeg: Integer; //Номер символа в начале таблицы
      TableBeg2: Integer; //Номер символа в начале данных таблицы
      Vr1,Vr2: OleVariant;
      F: Boolean;
      St: String;

      function ConvertString(S: String): String;
      {При формировании таблицы в качестве разделителя по умолчанию
      используется "-", который необходимо заменить на аналогичный
      символ с кодом #173}
      begin
        Result := StringReplace(S,'-',#173,[]);
      end;
    begin
      {Процедура экспортирует лишь те записи, у которых значение по-
следнего поля совпадает с FlagText. Если FlagText=," то экспортируют-
ся все записи}
      Vr1 := wdStory;
      //переход в конец документа

      Selection.EndKey(Vr1);

      //вставляется заголовок таблицы
      Document.Range.InsertAfter(Title);
      //настройки стиля абзаца
      Document.Paragraphs.Item(Document.Paragraphs.Count).Range.Select;
```

```
  Selection.ParagraphFormat.KeepWithNext := -1;
  Selection.ParagraphFormat.SpaceAfter := 14;
  //задается шрифт

  Selection.Font.Size := 15;
  WSelection.Font.bold := 1;
  //добавляется строка

  Document.Paragraphs.Add;
  Document.Paragraphs.Item(Document.Paragraphs.Count).Range.Select;
  Selection.ParagraphFormat.SpaceAfter := 0;
  Vr1 := wdStory;
  //переход в конец документа

  Selection.EndKey(Vr1);
  //запоминается положение курсора. Это начало будущей таблицы.
  //затем выбирается весь оставшийся текст, чтобы преобразовать
  //его в таблицу
  TableBeg := Selection.End;
  DataSet.First;
  //вставляются заголовки для всех видимых полей
  for I := 0 to DataSet.FieldCount-1 do
    if DataSet.Fields[i].Visible then
   Document.Range.InsertAfter (ConvertString(DataSet.Fields[I].
   DisplayLabel)+#9); Selection.EndKey(Vr1);
   //убираются последние символы табуляции
   {Символ табуляции используется в качестве разделителя для
столбцов таблицы}
  Selection.TypeBackspace;
  //применяется шрифт
  Document.Paragraphs.Item(Document.Paragraphs.Count).Range.Select;
  Selection.Font.Size := 14;
  Selection.Font.Italic := 1;
  Selection.Font.Bold := 0;
  //добавляется строка
  Document.Paragraphs.Add;
//Определение, были ли в таблице вообще записи для экспорта

  F := True;
//Строка для экспорта текста таблицы

  St := '';
```



```
  //начало данных в таблице

    TableBeg2 := Selection.End;

    if Dataset.RecordCount > 0 then begin
      repeat
        if (Dataset.fields[DataSet.Fields.Count-1].AsString = FlagText) or
(FlagText = '') then begin
          for I := 0 to DataSet.FieldCount-1 do
            if DataSet.Fields[I].Visible then
              //через табуляцию выводятся все видимые поля

              St := St+DataSet.Fields[I].AsString+#9;
          //убирается последний символ табуляции

          SetLength(St,Length(St)-1);
          //перенос строки

          St := St+#13;
          F := False;
        end;
        Dataset.Next;
      until Dataset.Eof;
    //переход в конец текста

      Selection.EndKey(Vr1);
      //вставка данных таблицы

      Selection.InsertAfter(ConvertString(St));
      //начало данных таблицы

      Vr1 := TableBeg2;
      //конец таблицы

      Vr2 := Selection.End;
      Selection.Font.Size := 12;
      Selection.Font.Bold := 0;
      Selection.Font.Italic := 0;
    end;
    //в том случае, если не экспортировалось ни одной записи
    //формируется пустая строка
    if F then begin
      for I := 0 to DataSet.FieldCount-1 do
```

```
        if DataSet.Fields[I].Visible then
          Document.Range.InsertAfter(' '+#9);
        Selection.EndKey(Vr1,EmptyParam);

        Selection.TypeBackspace;
      end;
      //начало будущей таблицы

      Vr1 := TableBeg;
      //конец будущей таблицы

      Vr2 :=  Selection.End;
      //выбирается диапазон

      Document.Range(Vr1,Vr2).Select;
      //преобразование в таблицу
      Selection.ConvertToTable;
      ColCount := 1;
      //задаются ширины колонок
      for I := 0 to DataSet.FieldCount-1 do
       if DataSet.Fields[I].Visible then begin
       Document.Tables.Item(Document.Tables.Count).Columns.Item(ColCount).
Width := DataSet.Fields[i].Tag;
        Inc(ColCount);
       end;
      //задаются границы
      Selection.Cells.Borders.Item (wdBorderLeft). LineStyle:=wdLineStyle-
Single;
      Selection.Cells.Borders.Item(wdBorderRight).LineStyle:= wdLi-
neStyleSingle;
      Selection.Cells.Borders.Item(wdBorderHorizontal).LineStyle:= wdLi-
neStyleSingle;
      Selection.Cells.Borders.Item(wdBorderTop).LineStyle:= wdLi-
neStyleSingle;
      Selection.Cells.Borders.Item(wdBorderBottom).LineStyle:= wdLi-
neStyleSingle;
      Selection.Cells.Borders.Item(wdBorderVertical).LineStyle:= wdLi-
neStyleSingle;
      Document.Paragraphs.Add;
      Document.Paragraphs.Item(Document.Paragraphs.Count-1).
Range.Select;
      Selection.ParagraphFormat.KeepWithNext:=0;
```



```
    end;

    end.
```

При нажатии на кнопку Button1 выполняется процедура Button1Click, в которой создаются объекты Word и Document, задаются титульный лист и колонтитулы, затем вызывается процедура TableExport – экспорта данных в таблицу Word. В данной программе использовалось позднее связывание. Чтобы сократить количество обращений к интерфейсам ссылки на объекты, Document и Select хранятся в отдельных переменных. По окончании работы все ссылки на объекты сервера автоматизации необходимо удалить, что достигается присвоением соответствующим переменным значения Null.

Задачи «Клиент-Сервер»

Во всех в вариантах задач требуется создать COM-сервер, предоставляющий указанный интерфейс, и реализовать приложение-клиент, использующее функции сервера:

1. Вычисление тригонометрических функций.
2. Калькулятор (поддерживающий сложение, вычитание, умножение, деление).
3. Обработка файла: нахождение длины файла (в символах), нахождение самого длинного слова.
4. Обработка файлов: сравнение двух файлов, нахождение одинаковых слов в файлах, объединение файлов.
5. Реализация файловых операций (открытие, чтение, запись и т. д.).
6. Сортировка массива (любым способом).
7. Обработка изображения (осветление, затемнение, установка прозрачности).
8. Кодирование изображения по заданному ключу (например, операцией XOR).
9. Шифрование текста методом XOR.
10. Работа со строкой текста: вычисление длины, нахождение подстроки, нахождение количества вхождений подстроки.
11. Обработка строки: перевод строки в верхний или нижний регистр.
12. Перевод чисел в текстовое представление (например, 111 → 'сто одиннадцать'). Числа могут быть до 1000000.

13. Перевод дат в различные текстовые представления.

14. Анализ html-документов. Необходимо обеспечить операции извлечения списка адресов (url) всех изображений, содержащихся в документе, всех ссылок и всех форм, а также извлечение текстового содержания документа (т. е. удаление всех тегов).

15. Поиск кратчайшего пути на графе. Предусмотреть возможность загрузки графа из текстового представления/файла (т. е. также требуется разработать формат текстового представления).

16. Перевод текста. Под переводом понимается простая замена слов. Также необходимо реализовать функции загрузки словарей.

17. Вычисление значений функций одной переменной, заданных в текстовой форме, например '=x^2+3'. Интерпретатор должен поддерживать как простые математические операции (+, –, *, /, ^), так и некоторый набор функций (Cos, Sin и т. д.).

18. Выполнить предыдущую задачу для функций нескольких переменных.

19. Хеш-таблица (для хранения чисел, записей, строк и т. п.).

20. Загрузка автомобилей. Есть список автомобилей и список грузов, требуется распределить груз между автомобилями и в качестве результата выдать список грузов для каждого автомобиля.

Список литературы

1. Архангельский А. Я. Программирование в Delphi 6 / А. Я. Архангельский. – М. : БИНОМ, 2002. – 1120 с.
2. Бокс Д. Сущность технологии COM. Библиотека программиста / Д. Бокс. – СПб. : Питер, 2001. – 400 с.
3. Лейнекер Р. COM+. Энциклопедия программиста / Р. Лейнекер. – М. : ДиаСофт, 2002. – 656 с.
4. Роджерсон Д. Основы COM / Д. Роджерсон ; пер. с англ. – 2-е изд., испр. и доп. – М. : Русская Редакция, 2000. – 400 с.
5. Рофэйл Э. COM и COM+. Полное руководство / Э. Рофэйл, Я. Шохауд ; пер. с англ. – Киев : ВЕК+, М. : Энтроп, 2000. – 560 с.
6. Тейксейра С. Delphi 4. Руководство разработчика / С. Тейксейра, К. Пачеко ; пер. с англ. – Спб. : Вильямс, 1999. – 912 с.
7. Хармон Э. Разработка COM-приложений в среде Delphi : учеб. пособие / Э. Хармон. – М. : Вильямс, 2000. – 464 с.

Ссылки в Internet

1. MSDN Library. Microsoft Corporation. – http://msdn.microsoft.com/library/en-us/fileio/base/file_management_functions.asp
2. Глущенко Ю. Экспорт из БД в Word. – http://www.delphimaster.ru/ articles/dbtoword/index.html.
3. Елманова Н. Автоматизация приложений Microsoft Office в примерах Часть 1. Microsoft Word и Microsoft Excel. – http://www.compress.ru/ Archive/CP/2000/11/25/.
4. Королевство Delphi. Виртуальный клуб программистов. http://www.delphikingdom.ru/
5. Статьи по программированию. – http://www.wasm.ru
6. Технологии программирования. – http://www.HiProg.com
7. Электронные книги и учебники по программированию. http://www.cnt.ru/~wh/x/books_n_manuals.html
8. Электронный каталог Научной библиотеки ВГУ. – http://www.lib.vsu.ru

Учебное издание

Артемов Михаил Анатольевич,
Вахтин Алексей Александрович,
Вощинская Гильда Эдгаровна,
Рудалев Валерий Геннадьевич

Основы COM-технологий

Учебно-методическое пособие для вузов

Подписано в печать 20.08.07. Формат 60×84/16. Усл. печ. л. 5.
Тираж 50 экз. Заказ 1765.

Издательско-полиграфический центр
Воронежского государственного университета.
394000, г. Воронеж, пл. им. Ленина, 10. Тел. 208-298, 598-026 (факс)
http://www.ppc.vsu.ru; e-mail: pp_center@typ.vsu.ru

Отпечатано в типографии Издательско-полиграфического центра
Воронежского государственного университета.
394000, г. Воронеж, ул. Пушкинская, 3. Тел. 204-133.

85