АКАДЕМИЯ СПЕЦИАЛЬНЫХ КУРСОВ ПО ИНФОРМАЦИОННЫМ ТЕХНОЛОГИЯМ
ASKit
IT-ОБУЧЕНИЕ ДЛЯ ПРЕДПРИЯТИЙ В САНКТ-ПЕТЕРБУРГЕ И РЕГИОНАХ
WWW.ASKIT.RU; info@ASKit.ru
САНКТ-ПЕТЕРБУРГ (812) 922-47-60

РОСТИСЛАВ МИХЕЕВ

VBA И ПРОГРАММИРОВАНИЕ В MS OFFICE ДЛЯ ПОЛЬЗОВАТЕЛЕЙ

СПЕЦИАЛЬНЫЙ КУРС

ЯЗЫК VBA

ОБЪЕКТНЫЕ МОДЕЛИ WORD, EXCEL, ACCESS, OUTLOOK, POWERPOINT, PROJECT

ВЗАИМОДЕЙСТВИЕ С БАЗАМИ ДАННЫХ

РЕШЕНИЕ РЕАЛЬНЫХ ЗАДАЧ ОТЕЧЕСТВЕННЫХ ПРЕДПРИЯТИЙ

Предисловие

Несколько лет назад автору — сертифицированному преподавателю Microsoft — поступил заказ: подготовить курс по программированию в Office. Задача была простая: существует группа из нескольких десятков человек, достаточно продвинутых пользователей, которые не имеют никакого опыта программирования. Одни пользователи занимались анализом трафика базовых станций в сети (заказчик был оператором сотовой связи российского масштаба), другие — проектами по развертыванию тех же базовых станций, третьи все это планировали и прогнозировали и т. п. И многие пользователи обращались к IT-подразделению с просьбой автоматизировать выполнение определенных задач, например:

- загрузку в Excel информации из базы данных SQL Server, дальнейший анализ (например, выявление тенденций) и представление результатов в стандартном виде;
- автоматическое создание сводных таблиц и графиков в Excel;
- проход по всем проектам (300—400 единиц) на Project Central (программное средство для корпоративной работы с проектами Microsoft Project) и замена в них каких-то элементов;
- создание стандартных документов Word, в которые бы подставлялись данные из базы данных.

И, конечно, список задач этим не ограничивался.

На этом предприятии были очень квалифицированные, но постоянно занятые программисты. Конечно, они отзывались на просьбы пользователей, но фактически намного больше времени уходило на постановку задачи, чем на ее решение. А через некоторое время задача вполне могла слегка измениться (например, нужно было ввести дополнительную ось на графике в Excel или поменять пару строк в шаблоне Word), и сотрудникам приходилось снова обращаться к разработчикам.

В результате мудрое руководство предприятия пришло к выводу, что проще научить пользователей автоматизировать свою работу самостоятельно, и поэтому был заказан этот курс.

Подготовка любого учебного курса — дело очень трудоемкое, и поэтому автор (у которого к тому времени был опыт преподавания более чем 30 официальных курсов Microsoft) постарался подобрать что-нибудь подходящее из официальных курсов Microsoft Official Curriculum (MOC). Однако его ждала неудача: у Microsoft были предусмотрены только курсы по Access различных версий плюс мини-курсы по очень специфическим вопросам: приложения коллективного использования (курс 2381), решения управления знаниями (курс 1904) и т. п. Сами курсы были скорее обзорными и предназначались для того, чтобы познакомить опытных разработчиков с новыми технологиями.

Тогда автор обратился к книжным полкам. Был куплен десяток книг, которые имели отношение к теме "Программирование в Office", но ни по одной из них сделать учебный курс было нельзя. Некоторые книги под программированием в Office понимали использование математических и финансовых функций в Excel, другие ограничивались рассмотрением элементарных программных конструкций VBA, в-третьих полкниги отводилось на объяснение основ объектно-ориентированного программирования (при этом в реальной работе на VBA определять пользовательские классы приходится достаточно редко). И ни одна из книг не давала возможность "увидеть за деревьями лес" — получить целостное представление об объектных моделях Word, Excel, Access, PowerPoint, Outlook и Project, чтобы дать возможность пользователю самостоятельно находить в них нужные объекты и создавать свои приложения.

Автору пришлось подготавливать этот курс самостоятельно. Был собран и прочитан весь возможный материал, скачано из Интернета и проанализировано несколько сотен приложений VBA, законспектирована официальная документация. Вместе с сотрудниками предприятий решались их проблемы, которые возникали на практике. Уже первый вариант курса был признан заказчиком (и его сотрудниками) очень удачным, а постепенно курс совершенствовался. Вместе с пользователями с разных предприятий на него иногда попадали (из любопытства) опытные разработчики, которые сами рассказывали немало интересного. Постепенно собирался новый материал. За счет общения с большим количеством слушателей удалось отобрать те моменты, которые наиболее важны в практической работе.

Результат — перед вами.

Эта книга отличается от многих других следующим:

- она "отлажена" на десятках слушателей с самых разных предприятий. Каждое замечание учитывалось и отражалось в курсе (а потом и в книге),

внимание акцентировалось на тех моментах, которые действительно важны для практической работы;

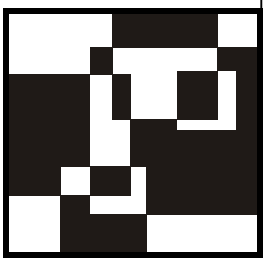
- учесть все многообразие ситуаций, которые возникают на предприятиях, невозможно. Поэтому в данной книге был сделан акцент не на рассмотрение отдельных случаев ("как вставить диаграмму в лист Excel"), а на том, как самостоятельно найти решение в подобной ситуации;
- для каждого приложения Office дается общая картина того, как устроена его объектная модель и из каких важнейших объектов она состоит (например, в Word это `Application` — `Document` — `Selection`, `Range` и `Bookmark`, в Excel `Application` — `Workbook` — `Worksheet` — `Range`). Знание этих программных объектов на 80% покрывает потребности при создании реальных приложений VBA;
- поскольку изначально эта книга была учебным курсом, было бы жаль не использовать некоторые преимущества, которые это дает. Для большинства глав предусмотрены задания для самостоятельной работы. Можно попытаться решить их самостоятельно, а можно использовать их просто как дополнительные примеры. После каждого задания приводится подробное решение с комментариями.

Несколько слов о технических моментах. Все примеры в книге приведены для приложений Microsoft Office 2003 в русскоязычной версии. Практически весь материал применим и к программированию в Office XP. Значительно больше отличий в Office 97 и в Office 2000, хотя основные моменты остаются неизменными во всех версиях. Несмотря на то, что в книге рассмотрены все встроенные функции текущей версии VBA и все главные объекты приложений Office текущей версии, сделан акцент на том, чтобы читатель понял, как можно самостоятельно найти необходимую информацию. Поэтому автор надеется, что книга пригодится и для работы с будущими версиями Microsoft Office, и с программными продуктами других фирм, в которых реализована поддержка языка VBA.

Автор совершенно не возражает против использования этой книги в качестве учебного пособия при проведении курсов в других учебных центрах или при организации обучения на предприятиях. Пользуйтесь на здоровье! И, если понравится, обращайтесь в нашу Академию за другими курсами. Наш адрес электронной почты — **info@askit.ru**, адрес Web-сайта — **www.askit.ru**. Мы с удовольствием проведем для вас курс из числа уже готовых или создадим новый учебный курс (в особенности по нестандартной тематике). И, может быть, из такого заказа возникнет новая книга — подобно тому, как появилась эта.

Course page:
http://www.askit.ru/custom/vba_office/vba_office_info.htm

ГЛАВА 1



Основы программирования в Microsoft Office

1.1. Зачем программировать в Microsoft Office

Ответ на этот вопрос прост: чтобы не делать лишней работы. Программирование в Office — это, прежде всего, уменьшение количества повторяющихся действий (и ручной работы, которая для этого требуется). Вот примеры некоторых типичных ситуаций, когда использование программирования просто напрашивается:

- вам с определенной периодичностью приходится изготавливать документы, очень похожие друг на друга: приказы, распоряжения в бухгалтерию, договоры, отчеты и т. п. Часто информацию можно взять из базы данных, тогда использование программирования может дать очень большой выигрыш во времени. Иногда данные приходится вводить вручную, но и тогда автоматизация дает выигрыш во времени и в снижении количества ошибок;
- разновидность такой же ситуации: одни и те же данные нужно использовать несколько раз. Например, вы заключаете договор с заказчиком. Одни и те же данные (наименование, адрес, расчетный счет, номер договора, дата заключения, сумма и т. п.) могут потребоваться во многих документах: самом договоре, счете, счете-фактуре, акте сдачи выполненных работ и т. д. Логично один раз ввести эту информацию (скорее всего, в базу данных), а затем автоматически формировать (например, в Word) требуемые документы;
- когда нужно сделать так, чтобы вводимые пользователем данные автоматически проверялись. Вероятность ошибки при ручном вводе данных зависит от многих факторов, но, согласно результатам некоторых исследо-

ваний, она в среднем составляет около 2%. "Вылавливать" потом такие ошибки в уже введенных данных — очень тяжелый труд, поэтому лучше сразу сделать так, чтобы они не возникали.

В общем, любое действие, которое вам приходится повторять несколько раз, — это возможный кандидат на автоматизацию. Например, занесение сотен контактов в Outlook, или замена ресурса в десятках проектов Project, или анализ информации из базы данных за разные периоды в таблице Excel — это те ситуации, когда знание объектных моделей приложений Office спасет вас от нескольких часов или даже дней скучного труда.

Конечно, есть еще практиканты и аналогичный бесплатный трудовой ресурс, но хочется ли вам потом заниматься еще и поиском ошибок за ними? Кроме того, программирование несет и другие преимущества для сотрудника, который использует его в работе:

- повышается авторитет сотрудника в глазах руководства и других коллег;
- если программы этого сотрудника активно используются на предприятии (им самим или другими работниками), то этим самым он защищает себя от сокращений, снижения зарплаты и т. п., ведь поддерживать и изменять программы в случае необходимости будет некому.

1.2. Что такое язык VBA

Поскольку эта книга предназначена для обычных пользователей, то без объяснения этого вопроса не обойтись. Формальное определение такое.

VBA (Visual Basic for Applications) — это диалект языка Visual Basic, расширяющий его возможности и предназначенный для работы с приложениями Microsoft Office и другими приложениями от Microsoft и третьих фирм.

В принципе, при программировании в Office можно вполне обойтись и без языка VBA. Подойдет любой COM-совместимый язык, например: обычный Visual Basic, VBScript, Java, JScript, C++, Delphi и т. п. Можно использовать и .NET-совместимые языки программирования: VB.NET, C# и т. п. Вам будут доступны все возможности объектных моделей приложений Office. Например, если сохранить следующий код в файле с расширением vbs и запустить его на выполнение, то будет запущен Word, в котором откроется новый документ и будет впечатан текст:

```
Dim oWord
Set oWord = CreateObject("Word.Application")
oWord.Visible = true
oWord.Documents.Add
oWord.Selection.TypeText ("Привет от VBScript")
```

Тем не менее, VBA — это обычно самый удобный язык для работы с приложениями Office. Главная причина проста — язык VBA встроен в приложения Office, и код на языке VBA можно хранить внутри документов приложений Office: в документах Word, книгах Excel, презентациях PowerPoint и т. п. Конечно же, этот код можно запускать из документов на выполнение, поскольку среда выполнения кода VBA (на программистском сленге — *хост*) встроена внутрь этих приложений.

В настоящее время VBA встроен:

- во все главные приложения Microsoft Office — Word, Excel, Access, PowerPoint, Outlook, FrontPage, InfoPath;
- в другие приложения Microsoft, такие как Visio и Project;
- в более 100 приложений третьих фирм, например, в CorelDRAW и CorelWordPerfect Office 2000, AutoCAD и т. п.

Но есть также и множество других преимуществ.

- VBA — универсальный язык. Освоив его, вы не только получите ключ ко всем возможностям приложений Office и других, перечисленных ранее, но и будете готовы к тому, чтобы:
  - создавать полноценные приложения на Visual Basic (поскольку эти языки — близкие родственники);
  - использовать все возможности языка VBScript (это вообще "урезанный" VBA). В результате в вашем распоряжении будут универсальные средства для создания скриптов администрирования Windows, Web-страниц (VBScript в Internet Explorer), Web-приложений ASP, для применения в пакетах DTS и заданиях на SQL Server, а также для создания серверных скриптов Exchange Server и многое-многое другое.
- VBA изначально был ориентирован на пользователей, а не на профессиональных программистов (хотя профессионалы пользуются им очень активно), поэтому создавать программы на нем можно быстро и легко. Кроме того, в Office встроены мощные средства, облегчающие работу пользователя: подсказки по объектам и по синтаксису, макрорекордер и т. п.
- При создании приложений на VBA вам, скорее всего, не придется заботиться об установке и настройке специальной среды программирования и наличии нужных библиотек на компьютере пользователя — Microsoft Office есть практически на любом компьютере.
- Несмотря на то, что часто приложения VBA выполняются медленнее, чем бы вам хотелось, они нересурсоемки и очень хорошо работают, например, на сервере терминалов. Но, как правило, для программ на VBA особых требований на производительность и не ставят: для написания игр, драй-

веров, серверных продуктов он не используется. По моему опыту, возникающие проблемы с производительностью VBA-приложений — это чаще всего не проблемы VBA, а проблемы баз данных, к которым они обращаются. Если проблемы действительно в VBA (обычно тогда, когда вам требуется сложная математика), то всегда есть возможность написать важный код на C++ и обращаться к нему как к обычной библиотеке DLL или встраиваемому приложению (Add-In) для Word, Excel, Access и т. п.

- Программы на VBA по умолчанию не компилируются, поэтому вносить в них исправления очень удобно. Не нужно разыскивать исходные коды и перекомпилировать программы.

В среде программистов-профессионалов считается, что быстрее всего научиться создавать профессиональные приложения можно именно при помощи VBA и объектов приложений Office. Другие языки программирования (C++, Java, Delphi) придется осваивать намного дольше, а их возможности во многом избыточны для большинства повседневных задач, которые встречаются на любом предприятии. Кроме того, использование возможностей объектов Office (графического интерфейса, средств работы с текстом, математических функций и т. п.) позволит резко снизить трудоемкость при создании приложений.

1.3. Макрорекордер: быстрое создание макросов

В большинство программ Microsoft Office (исключая Access и FrontPage) встроено замечательное средство, которое позволит вам создавать программы, вообще ничего не зная о программировании. Это средство называется макрорекордером.

Макрорекордер, как понятно из его названия, — это средство для записи макросов. *Макрос* — всего лишь еще одно название для VBA-программы, а макрорекордер — средство для его автоматического создания.

Внимание!

Приложения Microsoft Office 2003 по умолчанию настроены так, что не позволяют запускать макросы. Поэтому перед тем, как приступать к созданию макросов, в меню **Сервис | Макрос | Безопасность** переставьте переключатель **Уровень безопасности** в положение **Средняя** или **Низкая**, а потом закройте и снова откройте данное приложение. Это потребуется сделать только один раз в начале работы.

Принцип работы макрорекордера больше всего похож на принцип работы магнитофона: мы нажимаем на кнопку — начинается запись тех действий,

которые мы выполняем. Мы нажимаем на вторую кнопку — запись останавливается, и мы можем ее проиграть (т. е. повторно выполнить ту же последовательность действий).

Конечно, макрорекордер позволяет написать только самые простые VBA-программы. Однако и он может принести много пользы. Например, можно "положить" на горячие клавиши те слова, словосочетания, варианты оформления и т. п., которые вам часто приходится вводить (должность, название фирмы, продукт, ФИО директора и ответственного исполнителя и т. д.), этим вы сэкономите много времени.

Как показывает опыт, подавляющее большинство обычных пользователей и не подозревает о существовании макрорекордера, несмотря на то, что его применение позволило бы сэкономить им массу времени.

Перед созданием макроса в макрорекордере:

- ❒ необходимо очень тщательно спланировать макрос, хорошо продумав, что вы будете делать и в какой последовательности. Если есть возможность, определите подготовительные действия. Например, если нужно вставить текущую дату в начало документа, может быть, имеет смысл первой командой макроса сделать переход на начало документа (<Ctrl>+<Home>);
- ❒ посмотрите, нет ли готовой команды, которую можно сразу назначить клавише или кнопке на панели инструментов без создания макроса. Сделать это можно при помощи меню **Сервис | Настройка**. С вкладки **Команды** можно перетащить нужную команду на требуемую панель управления, и, нажав на этой же вкладке кнопку **Клавиатура**, в окне **Настройка клавиатуры** назначить для команды нужную комбинацию клавиш;
- ❒ если вы собираетесь при помощи макроса менять оформление текста, то правильнее вначале создать новый стиль с вашим оформлением, а потом уже применить этот стиль к тексту. В этом случае опять-таки можно обойтись без макроса, просто назначив стиль комбинации клавиш. Делается это при помощи того же диалогового окна **Настройка клавиатуры**, как и в предыдущем случае.

Чтобы создать макрос в макрорекордере (для тех программ Microsoft Office, для которых это средство предусмотрено, например, Word, Excel, PowerPoint, Project):

1. В меню **Сервис | Макрос** выберите команду **Начать запись**. В открывшемся окне **Запись макроса** (рис. 1.1) вам потребуется определить:
   - **Имя макроса**. Правило такое: имя не должно начинаться с цифры, не должно содержать пробелы и символы пунктуации. Максимальная длина в Excel — 64 символа, в Word — 80 символов. Можно писать по-русски;

- будет ли макрос назначен кнопке на панели управления или комбинации клавиш. Выполнить это, а также задать другие средства для вызова макроса можно и потом — об этом будет рассказано в следующем разделе;
- где сохранить макрос. В Word в вашем распоряжении текущий файл и шаблон для всех вновь создаваемых документов — Normal.dot, в Excel — текущая книга, возможность создать макрос одновременно с созданием новой книги и личная книга макросов PERSONAL.XLS (макросы из этой скрытой книги будут доступны во всех книгах). Подробнее про то, где может храниться программный код, мы поговорим в *разд. 2.2*;
- **Описание**. В это поле лучше ввести информацию о том, для каких целей создается этот макрос — это подарок не только для других пользователей, но и для себя (через несколько месяцев).

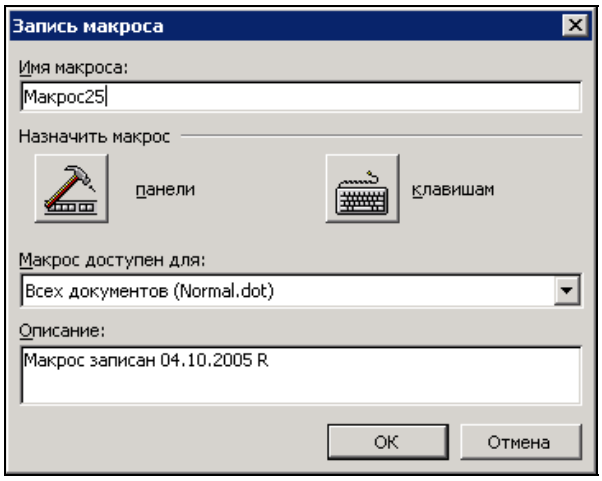


Рис. 1.1. Диалоговое окно **Запись макроса**

2. После нажатия кнопки **OK** или назначения кнопки или клавиатурной комбинации начнется запись макроса. Указатель мыши при этом примет вид магнитофонной кассеты и появится маленькая панель **Остановить запись**. На ней всего две кнопки — **Остановить запись** и **Пауза**. Если вы случайно закрыли эту панель, остановить запись можно через меню **Сервис | Макрос | Остановить запись**.
3. Самый простой способ запустить макрос, которому не назначена кнопка или клавиатурная комбинация, — в меню **Сервис** выбрать **Макрос | Макросы** (или нажать комбинацию клавиш <Alt>+<F8>), в открывшемся окне

Макрос (см. рис. 1.2) в списке выбрать нужный макрос и нажать кнопку **Выполнить**. Из этого же окна можно просматривать и редактировать макросы, удалять или перемещать их и т. п.

Если макросов создано много, то получить список всех назначений клавиш (включая назначения для встроенных макросов Word) можно при помощи меню **Сервис | Макрос | Макросы**, затем в окне **Макрос** в списке **Макросы из** выбрать **Команд Word**, а в списке **Имя** выбрать макрос **ListCommands** и нажать кнопку **Выполнить**. В ответ на приглашение нужно выбрать **Текущие настройки меню и клавиш** (иначе будет выведен полный список команд Word на 26 страниц). В ваш документ будет вставлена таблица с текущими назначениями клавиш, которую можно распечатать.

Если у вас уже есть значительное количество созданных при помощи макрорекордера макросов, то после освоения языка VBA имеет смысл подумать над ними и, может быть, внести изменения. Чаще всего стоит обратить внимание на следующие моменты:

- если в вашем макросе повторяются какие-либо действия, возможно стоит организовать цикл;
- может быть, есть смысл в ходе выполнения уточнить что-либо у пользователя (при помощи встроенной функции VBA `InputBox()` или элементов управления);
- чтобы в ходе выполнения макроса не возникало ошибок, можно реализовать в нем проверку текущих условий.

Как все это сделать, будет рассказано в следующих главах.

И еще один очень важный момент, связанный с макрорекордером. Помимо того, что он позволяет создавать простенькие программы, пригодные для самостоятельного использования без всяких доработок, макрорекордер — это еще и ваш разведчик в мире объектных моделей приложений Office. Опытные разработчики часто пользуются им для того, чтобы понять, какие объекты из огромных объектных моделей приложений Office можно использовать для выполнения тех или иных действий.

Приведу пример: вам нужно автоматизировать создание диаграмм в Excel. Поскольку в русской версии Excel для создания диаграммы вручную вы используете команду **Вставка | Диаграмма**, то, скорее всего, в справке по VBA вы начнете в первую очередь искать объект `Diagram`. И вы его найдете и, возможно, потратите определенное время на его изучение, прежде чем поймете, что это не та диаграмма! Объект `Diagram` представляет то, что в русской версии Excel называется "Схематическая диаграмма" (доступна из того же меню **Вставка**), а обычная диаграмма — это объект `Chart`. А вот если бы вы пустили вперед разведчика (т. е. создали бы диаграмму с записью в макрорекорде-

ре и посмотрели бы созданный код), он бы сразу указал нам нужное направление движения.

1.4. Четыре способа запуска макроса

Предположим, что макрос уже создан (в макрорекордере, как вы уже умеете, или средствами редактора Visual Basic, который вам предстоит освоить), и вы хотите или выполнить его один раз, или настроить возможность вызывать его постоянно. В нашем распоряжении — множество разных способов.

Самый простой, но и самый неудобный способ — воспользоваться окном **Макрос**, которое можно открыть при помощи меню **Сервис | Макрос | Макросы** (рис. 1.2).

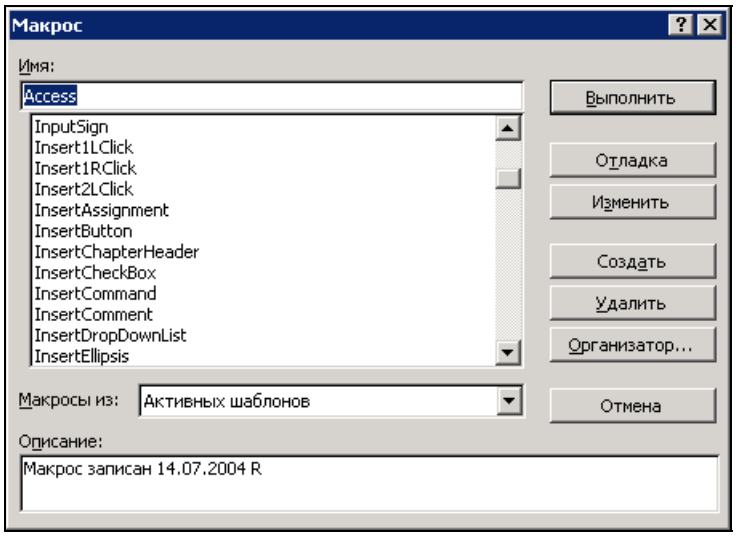


Рис. 1.2. Диалоговое окно **Макросы**

Из этого окна с помощью кнопок можно:

- **Выполнить** — запустить макрос на выполнение;
- **Отладка** — открыть макрос в редакторе Visual Basic и начать его пошаговое выполнение;
- **Изменить** — просто открыть макрос в редакторе Visual Basic;
- **Создать** — необходимо будет ввести имя создаваемого макроса и в редакторе Visual Basic будет автоматически создана процедура с определенным вами именем;

- **Удалить**;
- **Организатор** — поменять описание и назначенное сочетание клавиш.

Каждый раз открывать это окно, находить нужный макрос (а их может быть, например, несколько десятков) и нажимать на кнопку **Выполнить** — не самый быстрый вариант. Вряд ли он очень понравится вашим пользователям, да и вам самим работать будет неудобно. Поэтому в вашем распоряжении несколько более удобных вариантов.

Если вы пользуетесь макросом постоянно, то можно использовать самый быстрый способ его вызова — клавиатурную комбинацию. Например, сейчас, когда я пишу эту книгу, я "положил" на клавиатурные комбинации простые макросы, которые вводят нужный мне текст. Если мне нужно набрать "Visual Basic", я нажимаю <Alt>+<V>, если Microsoft Office — <Alt>+<M>. На клавиши (правда, уже без макросов) у меня разложены и все стили — заголовки, маркированные списки и т. п. Очень удобно!

На практике клавиатурным комбинациям есть смысл назначать только те макросы, которые используются каждый день, например, ввод информации об ответственном исполнителе, о руководителе, которому пойдет документ на подпись, о полном названии вашей организации и т. п. Главное — чтобы вы использовали их постоянно, иначе вы просто забудете, какое сочетание клавиш за что отвечает.

Назначить сочетание клавиш макросу можно очень просто.

В Word это выглядит так: с помощью меню **Сервис | Настройка** открываем одноименное окно и переходим на вкладку **Команды**. Затем нажимаем на кнопку **Клавиатура**: откроется окно **Настройка клавиатуры** (рис. 1.3).

В списке **Категории** нужно выбрать **Макросы**, в списке **Команды** — созданный вами макрос, установить указатель ввода в поле **Новое сочетание клавиш** и нажать требуемое сочетание клавиш. Помимо обычных сочетаний типа <Alt>+<1>, <Alt>+<M> и т. п., можно использовать и более сложные. Например, вы вставляете при помощи макросов два варианта названия вашей организации: полное и краткое. Этим макросам можно назначить клавиатурные комбинации вида <Alt>+<N>, <F> и <Alt>+<N>, <S>. Это значит, что если вы вначале нажмете вместе клавиши <Alt>+<N>, а затем <F>, то соответствующий макрос будет выполнен. Вводить такое сочетание клавиш в поле **Новое сочетание клавиш** нужно точно так же, как вы будете его применять.

После того как нужное сочетание клавиш будет введено, нажмите кнопку **Назначить**, а потом **Закрыть**.

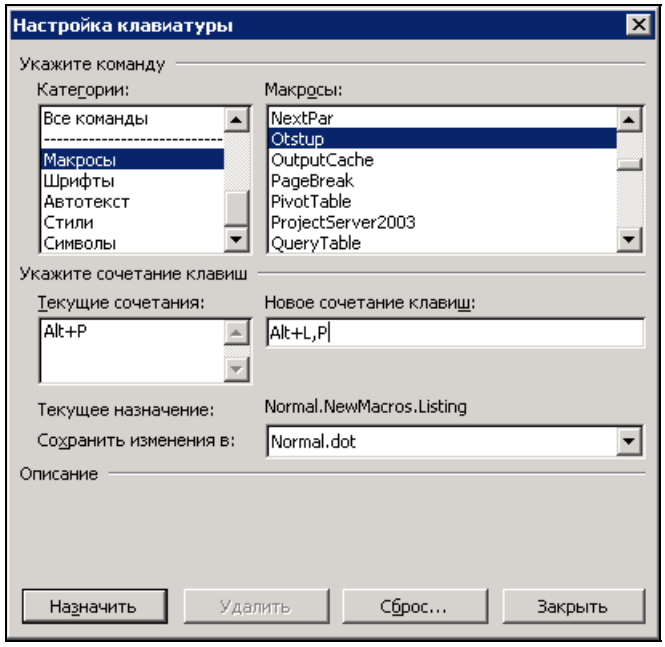


Рис. 1.3. Окно настройки клавиатурных комбинаций

Внимание!

Следите за надписью **Текущее назначение** в этом диалоговом окне. Вполне возможно, что выбранному вами сочетанию клавиш уже назначен другой макрос или встроенная команда. Если вы проигнорируете это сообщение, то вы переназначите эту комбинацию вашему макросу. Но если пользователь привык использовать эти клавиши для других целей (<Ctrl>+<S>, <Ctrl>+<N> и т. п.), он может быть очень недоволен.

В Excel кнопки **Клавиатура** в окне **Настройка** (меню **Сервис | Настройка**) вы не найдете. Здесь придется назначать клавиатурные комбинации по-другому: в меню **Сервис** выбрать **Макрос | Макросы**, выбрать в списке нужный макрос и нажать кнопку **Параметры**. Откроется окно **Параметры макроса** (рис. 1.4), в котором вы сможете выбрать нужную клавиатурную комбинацию (только в сочетании с клавишей <Ctrl>) и ввести описание макроса. Вообще говоря, любое сочетание клавиш можно назначить макросу и в Excel, но простыми способами этого сделать нельзя — придется писать программный код, в котором будут перехватываться события приложения.

Как мы уже говорили, клавиатурные комбинации есть смысл назначать только тем макросам, которыми вы пользуетесь каждый день. А что делать с полезными макросами, которые активно используются, к примеру, в отчетный

период, а потом опять ждут своего часа целый месяц? Подавляющее большинство пользователей за этот месяц забывает все назначенные клавиши и теряет те бумажки, на которых вы им записали эти клавиатурные комбинации. Да и сам разработчик (у меня это случается регулярно) вполне может забыть, что именно нужно нажимать для запуска старого макроса.

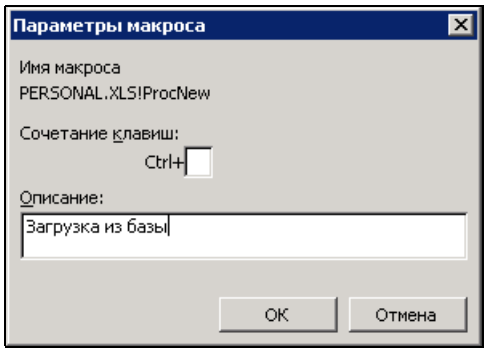


Рис. 1.4. Диалоговое окно **Параметры макроса**

Лучший выход в такой ситуации — назначить макрос пункту меню или кнопке на панели управления. Пожалуй, назначение пункту меню даже лучше — больше возможностей упорядочить макросы и использовать понятные названия. Однако нажимать кнопки на панели инструментов быстрее, так что выбирайте сами, что вам больше нравится. Создание и настройка новой панели инструментов для вызова макросов в Word может выглядеть так:

1. В меню **Сервис** выберите **Настройка** и перейдите на вкладку **Панели инструментов**.
2. Нажмите кнопку **Создать**, введите название панели (например, **Мои_макросы**) и выберите тот документ, в котором она будет создана. Если вы выберете Normal.dot, то панель будет доступна для всех документов Word на этом компьютере (что чаще всего и необходимо). Другой вариант — создать панель инструментов в том документе Word, который у вас открыт. В этом случае панель будет доступна только из этого файла.
3. После того как вы нажмете кнопки **OK** и **Закрыть**, будет создана новая пустая панель (которая будет находиться где-нибудь прямо поверх документа). Чтобы было удобней, перетащите ее к стандартным панелям инструментов, а потом вновь воспользуйтесь командой главного меню **Сервис | Настройка**. В окне **Настройка** перейдите на вкладку **Команды**, в списке **Категории** выберите **Макросы** и просто перетащите на панель инструментов нужные макросы из списка **Команды**. Если на панель инструментов нужно поместить не один, а несколько макросов, то, возможно,

удобнее будет нажать кнопку **Упорядочить команды** и воспользоваться очень удобным диалоговым окном **Изменение порядка команд** (рис. 1.5).

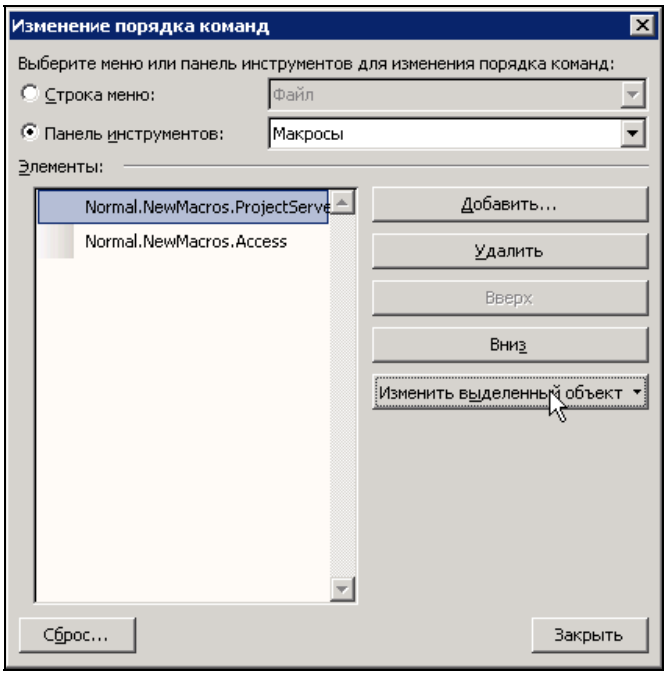


Рис. 1.5. Диалоговое окно **Изменение порядка команд**

Мы добавили нужные кнопки на панели инструментов, но пока они выглядят не очень интересно (например, **Normal.NewMacros.QueryTable**). Вряд ли такое название что-то скажет пользователю. Поэтому следующее действие — настройка кнопок. Для этого при открытом окне **Настройка** (это условие обязательно!) просто щелкните правой кнопкой мыши по кнопке панели инструментов, которую надо настроить. Откроется специальное контекстное меню (рис. 1.6).

С помощью этого меню можно:

- **Удалить** — удалить кнопку (также можно просто перетащить ее обратно с панели на окно **Настройка**);
- **Имя** — ввести имя, т. е. надпись на кнопке или пункте меню. Для меню использование надписи удобно, для кнопки на панели инструментов — не очень, поскольку это занимает много места;
- **Копировать значок на кнопке** и **Вставить значок для кнопки** — воспользоваться понравившимся вам значком с другой кнопки;

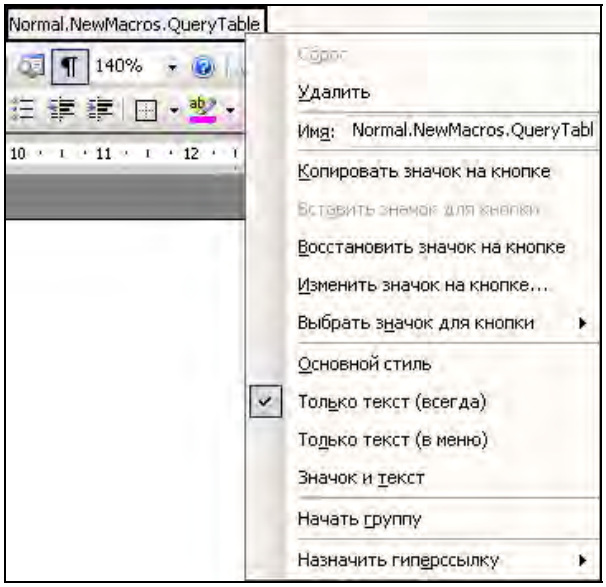


Рис. 1.6. Контекстное меню для настройки кнопки на панели инструментов

- **Изменить значок на кнопке** — откроется скромный редактор, в котором вы сможете сами нарисовать нужный значок;
- **Выбрать значок для кнопки** — выбрать один из 42 стандартных значков. На самом деле только в Word значков, которые можно использовать, несколько тысяч;
- **Основной стиль** — под этой надписью скрывается то, что нам обычно и нужно: чтобы кнопка была представлена только рисунком, безо всяких надписей;
- **Только текст (всегда)**, **Только текст (в меню)**, **Значок и текст** — определяют, что именно из набора надпись/рисунок будет показано на кнопке. Естественно, наиболее часто используемый вариант — **Основной стиль**;
- **Начать группу** — слева от кнопки появится вертикальная черта (разделитель);
- **Назначить гиперссылку** — назначить ссылку на другое место в вашем документе или на страницу в Интернете.

Конечно же, мы могли обойтись и без создания своей панели управления, просто добавив новые кнопки в существующие (точно таким же перетаскиванием). Аналогичным образом мы можем преобразовать стандартные панели инструментов. Однако не забывайте, что все эти преобразования доступны только при открытом диалоговом окне **Настройка**.

Создание меню производится немного по-другому:

1. Откройте то же диалоговое окно **Настройка** (меню **Сервис | Настройка**).
2. В списке **Категории** выберите **Новое меню**.
3. Перетащите команду **Новое меню** из списка **Команды** того же окошка (рис. 1.7) в нужное место основного меню.

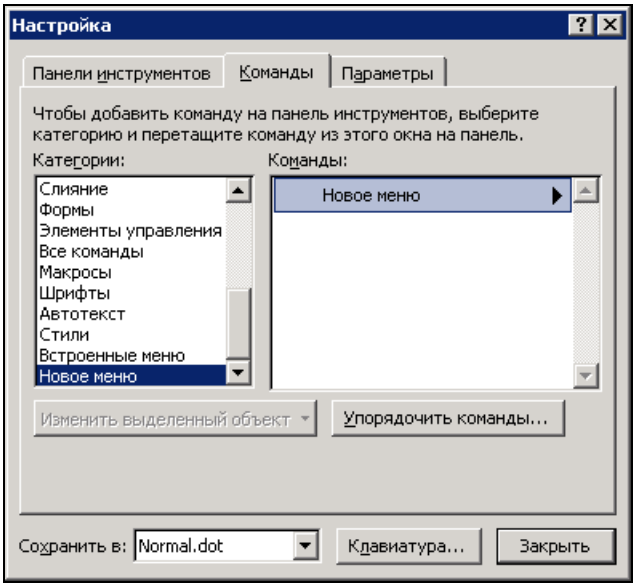


Рис. 1.7. Диалоговое окно **Настройка**

4. Далее точно так же при открытом окне **Настройка** щелкните правой кнопкой мыши по созданному вами пункту меню и переименуйте его (в нашем примере назовите его **Макросы**).

Далее нужно нажать кнопку **Упорядочить команды** в окне **Настройка**. В открывшемся диалоговом окне **Изменение порядка команд** (рис. 1.8) нужно в списке **Строка меню** выбрать **Макросы** и добавить в него нужные элементы (т. е. созданные вами макросы). Переименовать их можно при помощи кнопки **Изменить выделенный объект** прямо из этого окна.

В результате у вас может получиться очень милое меню, в котором пользователю запутаться будет трудно (рис. 1.9).

В Excel все очень похоже, но чуть-чуть по-другому. Если в Excel мы откроем окно **Настройка** (меню **Сервис | Настройка**) и в списке **Категории** выберем **Макросы**, то вместо списка макросов в списке **Команды** будет две возможности: **Настраиваемая команда меню** и **Настраиваемая кнопка** (рис. 1.10).

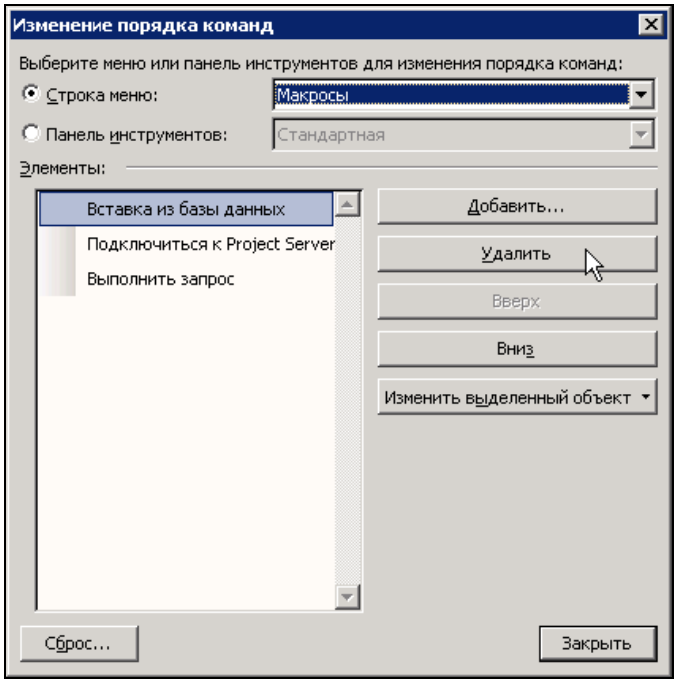


Рис. 1.8. Окно **Изменение порядка команд**

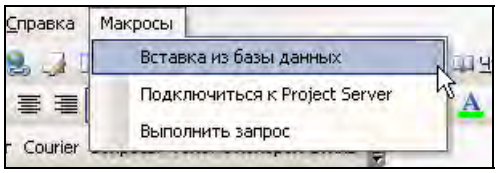


Рис. 1.9. Пример меню для запуска макросов

Настраиваемая кнопка — это готовая кнопка, которую можно перетащить на панель инструментов, а потом открыть для нее контекстное меню и воспользоваться командой **Назначить макрос**. Конечно же, для выбора иконки, формата отображения и т. п. можно воспользоваться и другими возможностями контекстного меню, которые доступны и в Word.

Для создания нового меню в Excel нужно точно так же создать новое меню, как и в Word, а потом нажать на кнопку **Упорядочить команды** и добавить в это меню несколько элементов **Настраиваемая команда меню**. Их реальная настройка (в том числе и назначение макросов) производится по нажатию кнопки **Изменить выделенный объект**.

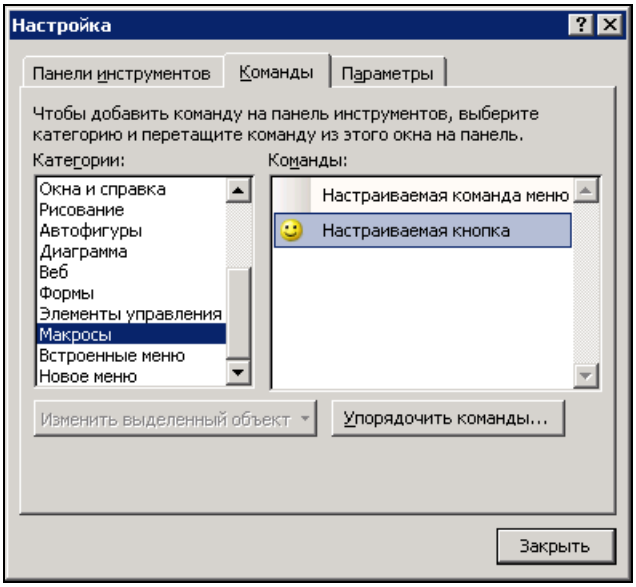


Рис. 1.10. Создание кнопки для запуска макроса в Excel

В подавляющем большинстве остальных приложений Office (PowerPoint, Project, Outlook и т. п.) работа с макросами происходит так же, как и в Word.

Есть еще один способ предоставить пользователю возможность запускать макросы — самый функциональный, но и самый трудоемкий: создать специальную графическую форму, на которую можно поместить, например, ниспадающий список макросов. При использовании этого способа можно предусмотреть дополнительные элементы управления для ввода параметров, которые макросы смогут "подхватывать" во время выполнения (напрямую параметры макросам передаваться не могут, поскольку макрос — это процедура, не принимающая параметров). Однако использование этого способа требует написания программного кода. Как работать с формами и элементами управления на них, будет рассказано в *гл. 5*. После этого создание такой формы не составит никакого труда.

Есть еще одна специальная возможность для запуска макросов: сделать так, чтобы они запускались при возникновении специального события. Таким событием может стать, например, внесение изменений на лист Excel, открытие книги Excel или документа Word и т. п. Подробнее про работу с событиями будет рассказано также в *гл. 5*. Однако можно обеспечить автоматический запуск макроса и без программирования: достаточно просто назначить ему специальное имя. Например, для Word список таких специальных названий представлен в табл. 1.1.

Таблица 1.1. *Специальные названия макросов для Word*

Имя процедуры	Когда запускается
`AutoExec`	При запуске Word (этот макрос должен храниться в Normal.dot)
`AutoNew`	При создании нового документа
`AutoOpen`	При открытии любого документа (если в Normal.dot) или при открытии документа, в котором находится макрос с таким именем
`AutoClose`	При закрытии документа
`AutoExit`	При выходе из Word

В Excel предусмотрены специальные имена макросов для рабочей книги `Auto_Open`, `Auto_Close`, `Auto_Activate` и `Auto_Deactivate`. Однако Microsoft предупреждает, что эти возможности оставлены только для обратной совместимости, и рекомендует пользоваться событийными процедурами.

Еще один момент, связанный с макросами `Auto`: поскольку раньше эти возможности активно использовались вирусами, в Office 2003 по умолчанию эти макросы запускаться не будут. Для того чтобы обеспечить им возможность запуска, необходимо изменить установленный уровень безопасности в меню **Сервис | Макрос | Безопасность** на **Низкий**.

Ну и последняя, по моему опыту, самая малоизвестная, но, тем не менее, очень полезная возможность для запуска макросов. Вы можете запустить их из командной строки при запуске Word или Excel, указав имя макроса в качестве параметра командной строки. Например, чтобы открыть Word и сразу выполнить макрос `MyMacros` из Normal.dot, можно воспользоваться командой:

```
winword.exe /mMyMacros
```

Очень удобно использовать эту возможность, если создать несколько ярлыков для запуска приложения Office, например, на рабочем столе, изменить в них командную строку и использовать для запуска приложения с одновременным запуском макросов.

Задание для самостоятельной работы 1: Запись макроса для автоматического ввода текста в Word

Ситуация:

Вам несколько раз в день необходимо передавать распоряжения в бухгалтерию. Каждое распоряжение должно заканчиваться строками, аналогичными представленным на рис. 1.11.

Генеральный директор:	Иванов А. А.
Отв. исполнитель Петрова М. М. т. 55-55	

Рис. 1.11. Строки, ввод которых нужно автоматизировать

ЗАДАНИЕ:

Напишите при помощи макрорекордера макрос, который бы автоматически создавал такие строки (вместо "Петрова М. М." подставьте ваши данные).

Созданный макрос должен быть доступен для всех создаваемых вами документов. Он должен запускаться по нажатию кнопки с рожицей (рис. 1.12).

Создайте новый документ, запустите макрос на выполнение и убедитесь, что он работает.

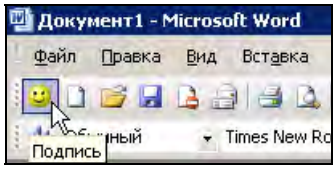
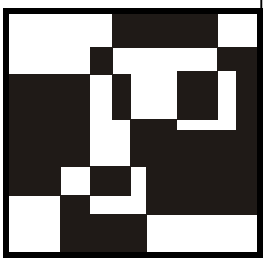


Рис. 1.12. Веселая кнопка для пользователя

Ответ к заданию 1

1. Откройте новый документ в Word. В меню **Сервис** выберите **Макрос | Начать запись**. В окне **Запись макроса** в поле **Имя макроса** введите "Подпись" (без кавычек), убедитесь, что в поле **Макрос доступен для** стоит значение **Всех документов (Normal.dot)**, и нажмите кнопку **Назначить макрос панели**.
2. В окне **Настройка** на вкладке **Команды** перетащите элемент **Normal.NewMacros.Подпись** в нужное место на панели управления. Затем щелкните по перемещенному элементу правой кнопкой мыши, в контекстном меню выберите пункт **Выбрать значок для кнопки**, затем выберите изображение улыбающейся рожицы. Еще раз щелкните правой кнопкой мыши по этому элементу и в контекстном меню выберите **Основной стиль**. Нажмите на кнопку **Закрыть** окна **Настройка**. Начнется запись макроса.
3. Введите нужный текст, а затем нажмите на кнопку **Остановить запись** (или в меню **Сервис | Макрос** выберите команду **Остановить запись**).
4. Создайте новый документ Word и убедитесь, что новая кнопка работает и там.

ГЛАВА 2



Знакомство с редактором Visual Basic

2.1. Общие сведения

Во многих ситуациях макрорекордер очень удобен, но в реальной работе одним им обойтись невозможно. Слишком много не умеет делать макрорекордер: он не умеет проверять значения, чтобы в зависимости от этого выполнять какое-либо действие, не работает с циклами, не умеет перехватывать и обрабатывать ошибки. Он использует только ограниченный и не лучший набор объектов (например, при вводе текста в Word использует чувствительный к действиям пользователя объект `Selection` вместо более надежного объекта `Range`). Макросы, которые созданы в макрорекордере, очень ограничены с функциональной точки зрения.

Полные возможности программирования в Office раскрываются при использовании редактора Visual Basic, и при серьезной работе без него не обойтись. В то же время, по моим наблюдениям, большинству обычных пользователей самостоятельно разобраться во всех способах его использования очень сложно.

В этой главе мы рассмотрим возможности редактора Visual Basic и познакомимся с тем, что можно сделать с помощью его многочисленных окон.

Внимание!

Для создания программ, которые используют возможности объектных моделей приложений Microsoft Office, использовать язык VBA и редактор Visual Basic совсем не обязательно. Вы вполне можете создавать такие программы на любом COM-совместимом языке, например, обычном Visual Basic, C++, Delphi, Java, VBScript и JavaScript, ActivePerl, C#, Visual Basic .NET и т. п. Однако язык VBA (и редактор Visual Basic) изначально создавались для автоматизации приложений Microsoft Office, и работать в них с объектными моделями приложений Microsoft Office удобнее всего.

Прежде чем начать работать с редактором Visual Basic, нужно его открыть. Во всех приложениях Office это делается одинаково:

- самый простой способ — в меню **Сервис | Макрос** выбрать **Редактор Visual Basic**;
- самый быстрый способ — нажать клавиши <Alt>+<F11>;
- можно также воспользоваться кнопкой на панели инструментов **Visual Basic** (предварительно сделав ее видимой);
- можно вызвать редактор при возникновении ошибки в макросе;
- можно открыть готовый макрос для редактирования в диалоговом окне **Макрос**.

В любом случае откроется окно, похожее на представленное на рис. 2.1.

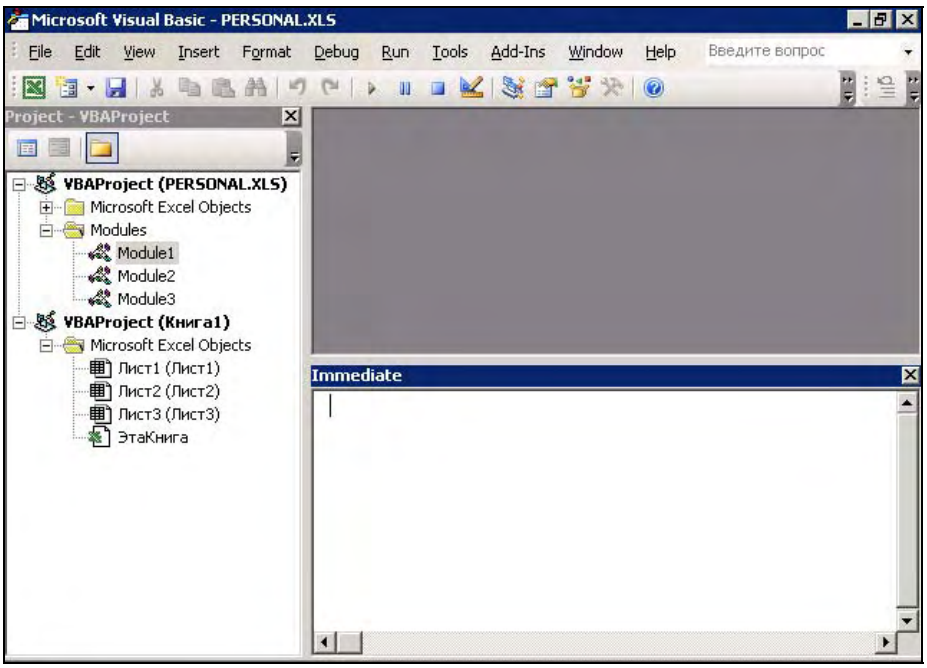


Рис. 2.1. Окно редактора Visual Basic в Excel

Внимание!

В окне редактора Visual Basic можно работать одновременно с работой в приложении, откуда этот редактор был вызван. Переход между окнами осуществляется через <Alt>+<Tab> (в редактор также можно "прыгнуть", повторно нажав <Alt>+<F11>).

Всего в редакторе Visual Basic предусмотрено 9 дополнительных окон:

- **Project Explorer** — окно проводника проекта. По умолчанию оно открыто и находится в левой части окна редактора Visual Basic. В нем можно просмотреть компоненты проекта и выполнить множество операций. Подробнее о работе в этом окне — в *разд. 2.2*;
- **UserForm** — окно формы. Появляется тогда, когда вы редактируете пользовательскую форму при помощи дизайнера форм. Подробнее про пользовательские формы и работу с ними — в *гл. 5*;
- **Toolbox** — панель инструментов управления. Из нее можно добавить элементы управления в форму или в сам документ. Появляется вместе с окном формы и будет рассматриваться тоже в *гл. 5*;
- **Properties** — одно из самых важных окон. Через него можно просмотреть свойства элемента управления или компонента проекта и изменить их;
- **Code** — окно программного кода. В этом окне выполняется основная работа по написанию кода макроса. При открытии программного модуля открывается автоматически. Приемы работы с этим окном будут рассмотрены в *разд. 2.3*;
- **Object Browser** — обозреватель объектов. Необходим для получения информации о классах, доступных программе. Подробная информация — в *гл. 4*, которая посвящена работе с объектами;
- **Watch** — окно контролируемых выражений. Используется во время отладки для отслеживания значений выбранных переменных программы и выражений. Работа в этом окне так же, как и работа с окнами **Locals** и **Immediate**, будет рассмотрена в *гл. 6*, в которой рассказывается про перехват ошибок и отладку;
- **Locals** — окно локальных переменных. Нужно для отслеживания во время отладки значений переменных текущей процедуры;
- **Immediate** — окно для немедленного выполнения команд в ходе отладки. Оно позволяет выполнить отдельные строки программного кода и немедленно получить результат.

Найти какое-либо окно можно очень просто: нужно выбрать в меню **View** одноименную команду, и если окно было скрыто, оно появится в редакторе.

И еще один момент, который многих пользователей может разочаровать. В русских версиях приложений Office для редактора Visual Basic предусмотрен англоязычный интерфейс. Справка по языку VBA и объектным моделям приложений Office — тоже только на английском. К сожалению, русифицированных вариантов не существует. Однако, по моим наблюдениям, знание

английского языка для того, чтобы писать программы в VBA, не критично (хотя и очень полезно): программы вполне можно создавать, не зная английского.

2.2. Окно проводника проекта (*Project Explorer*) и структура проекта VBA

Окно проводника проекта при первой активизации редактора Visual Basic обычно открыто. Если оно случайно было закрыто, то вызвать его можно тремя способами:

- нажать клавиши <Ctrl>+<R>;
- нажать кнопку **Project Explorer** на панели инструментов **Standard**;
- воспользоваться меню **View | Project Explorer**.

В окне **Project Explorer** представлено дерево компонентов вашего приложения VBA.

Самый верхний уровень — это *проект (Project)*, которому соответствует документ Word, рабочая книга Excel, презентация PowerPoint или другой файл, с которым работает данное приложение. Например, если вы открыли редактор Visual Basic из Word, то в **Project Explorer** будут представлены все открытые в настоящее время файлы Word и шаблон Normal.dot. Если редактор Visual Basic открыт из Excel, то в **Project Explorer** будут открытые книги Excel и специальная скрытая книга PERSONAL.XLS.

Кроме того, что обычно содержится в документах Office (текст, рисунки, формулы и т. п.), каждый проект (который и является документом) — это одновременно и контейнер для хранения стандартных модулей, модулей классов и пользовательских форм. Добавить в проект каждый из этих компонентов можно при помощи меню **Insert** или через контекстное меню в **Project Explorer**.

Стандартный модуль — это просто блок с текстовым представлением команд VBA. В модуле этого типа может быть только два раздела:

- раздел объявлений уровня модуля (объявление переменных и констант уровня модуля);
- раздел методов модуля (расположение процедур и функций).

При работе макрорекордера в Word в проекте Normal.dot или в текущем документе (в зависимости от места сохранения макроса) автоматически создается стандартный модуль **NewMacros** (в Excel — **Module1**), куда и записываются все создаваемые макрорекордером макросы.

В большинстве проектов VBA используется только один стандартный модуль, куда записывается весь код. Создавать новые стандартные модули имеет смысл только из следующих соображений:

- для удобства экспорта и импорта (из контекстного меню в **Project Explorer**). Так можно очень удобно обмениваться блоками кода между приложениями VBA (и обычного VB);
- для повышения производительности. При вызове любой процедуры модуля происходит компиляция всего модуля, поэтому иногда выгоднее разместить процедуры в разных модулях, чтобы компилировать только нужный в данный момент код;
- для улучшения читаемости. Если ваше приложение выполняет разные группы задач, то код, относящийся к каждой группе, лучше поместить в свой модуль.

Модули классов позволяют создавать свои собственные классы — чертежи, по которым можно создавать свои объекты. Обычно модули классов используются только в очень сложных приложениях. В обычных приложениях VBA они применяются редко, и в этой книге рассматриваться не будут.

Пользовательская форма является одновременно хранилищем элементов управления и программного кода, который относится к ним, к самой форме и происходящими с ними событиями. Подробнее про формы будет рассказано в *гл. 5*.

Еще один важный контейнер в **Project Explorer** — *контейнер References*, т. е. контейнер ссылок (в Excel его нет). В нем показывается, ссылки на какие другие проекты (документы Word) есть в нашем проекте, и, соответственно, какие "чужие" программные модули мы можем использовать. По умолчанию в каждый проект Word помещается ссылка на **Normal** (т. е. на шаблон Normal.dot), и вы в любом файле можете использовать макросы этого шаблона.

Обратите внимание, что в этом контейнере хранятся только ссылки на другие документы. Про добавление ссылок на другие объектные библиотеки мы поговорим в *гл. 4*.

Еще одна полезная возможность **Project Explorer** — *настройка свойства проекта*. Для этого нужно щелкнуть правой кнопкой мыши по узлу **Project** (**VBAProject** в Excel) и в контекстном меню выбрать **Project Properties** (окно свойств проекта можно открыть и через меню **Tools | Project Properties**). В этом окне можно:

- изменить имя проекта. Это может потребоваться, если у вас есть ссылки на проект с таким же именем;

- ввести описание проекта, информацию о файле справки и параметры, которые будут использоваться компилятором;
- защитить проект, введя пароль. Не зная этот пароль, проект нельзя будет просмотреть или отредактировать.

Тем не менее в окне **Project Explorer** обычно приходится выполнять следующие действия.

- Если вам нужно создать свой макрос вручную, а макросов в данном документе еще нет, то нужно щелкнуть правой кнопкой мыши по узлу проекта (строке, выделенной полужирным шрифтом) и в контекстном меню выбрать команду **Insert | Module**. В проекте будет создан новый модуль и сразу открыт в окне редактора кода. Что делать в этом окне — об этом в следующем *разд. 2.3*.
- Если вы уже создавали макросы в этом проекте (макрорекордером или вручную), то модуль будет уже создан. Его можно увидеть под контейнером **Modules**. Чтобы его открыть в окне редактора кода, достаточно щелкнуть по модулю два раза левой кнопкой мыши. Там можно будет найти макросы, созданные вами ранее средствами макрорекордера.

Внимание!

Обязательно подумайте, где вам будет нужен создаваемый код — только в одном документе или во всех документах данного приложения. Если он будет нужен только в одном документе, используйте стандартный программный модуль этого документа. Если во всех, то используйте программные модули проекта **Normal** (в Word) или PERSONAL.XLS (в Excel).

- Если вам нужно создать графическую форму с элементами управления (кнопками, текстовыми полями, раскрывающимися списками и т. п.), то нужно щелкнуть правой кнопкой мыши по узлу проекта и в контекстном меню выбрать **Insert | UserForm**. Новая форма будет создана и открыта в режиме дизайнера форм. Подробнее о работе с этим редактором — в *гл. 5*.

Теперь, когда программный модуль создан (или найден), можно приступать к работе с редактором кода VBA.

2.3. Работа с редактором кода (*Code Editor*)

2.3.1. Как открыть редактор кода и как он устроен

В редакторе кода выполняется основная часть работы по программированию, поэтому знать приемы работы с ним нужно очень хорошо. Открыть окно редактора кода можно несколькими способами:

- дважды щелкнуть по объекту модуля в **Project Explorer** (или выделить его и нажать клавишу <Enter>);

- выбрать нужный элемент (в **Project Explorer**, в дизайнере форм и т. п.) и в контекстном меню выбрать **View Code**;
- выделить нужный элемент и нажать клавишу <F7> (альтернатива — команда меню **View | Code**).

Редактор программного кода — это, по сути, обычный текстовый редактор, и в нем вы можете вырезать и вставлять код, перетаскивать фрагменты кода, скопировать путем перетаскивания с нажатой клавишей <Ctrl> — в вашем распоряжении почти все те же возможности, что и в редакторе Word. Однако он все-таки предназначен для специализированной задачи — создания кода программы. О его специальных возможностях рассказывается далее.

2.3.2. Список объектов и список событий

В верхней части окна редактора кода находятся два раскрывающихся списка. Слева находится *список объектов*. В нем вы можете выбрать объект, к которому будет относиться ваш код. Если вы открыли программный код модуля, то здесь будет только пункт **General**. Другое дело, если открыта форма, то в этом списке вы сможете выбрать саму форму или любой ее элемент управления и записать для него код.

Список справа — это *список процедур и событий*. В нем есть раздел **Declarations** — объявления уровня всего модуля и список всех процедур (макросов) для стандартного модуля или событий, если создается код для формы. При выборе нужного события будет автоматически создана нужная процедура, обрабатывающая это событие.

2.3.3. Закладки и разделение окна редактирования

Иногда в процессе написания программного кода в одном месте вам в голову приходит идея, относящаяся к другой части кода. Хочется перепрыгнуть в другое место, но разыскивать потом ту строку, где была прервана работа, очень не хочется. В этом случае опытные программисты используют закладки. *Закладка* (как и в случае с обычной книгой) — это метка, при помощи которой можно быстро найти нужное место. Работа с закладками производится либо с панели инструментов **Edit** (ее вначале нужно сделать видимой), либо через меню **Edit | Bookmark**. Для того чтобы включить или отключить закладку, нужно установить указатель ввода на нужную строку и воспользоваться командой **Toggle Bookmark**.

Часто бывает очень удобным разделение окна редактирования на две части — для просмотра разных частей модуля, для копирования и т. п. Делается это при помощи линии разбивки — маленького бегунка над вертикальной полосой прокрутки.

2.3.4. Как редактор помогает писать код

В редактор кода встроено множество средств, которые облегчают жизнь разработчику. Рассмотрим самые важные из них.

Самое полезное средство — это *получение списка свойств и методов*. В большинстве VBA-программ используются свойства и методы различных объектов (подробнее об этом — в *гл. 4*), при этом многие методы принимают параметры. Помнить точное название каждого свойства и метода, а также очередность передачи параметров невозможно, а разыскивать в очередной раз справку по этому объекту в документации — непроизводительная трата времени.

Показ списка свойств и методов в редакторе Visual Basic включен по умолчанию. Пользоваться этой возможностью очень просто: достаточно напечатать имя переменной, представляющей объект, и поставить после него точку. Автоматически откроется список всех свойств и методов этого объекта. В этом списке можно выбрать нужное свойство или метод (клавишами со стрелками или мышью, а если список большой, то можно набрать первые буквы имени свойства или метода), а затем нажать клавишу <Tab>. Если вы случайно закрыли список, то открыть его заново можно при помощи меню **Edit | List Properties/Methods** или клавиш <Ctrl>+<J>.

Если показ списка свойств и методов у вас отключен, то включить его можно при помощи меню **Tools | Options** (флажок **Auto List Members** на вкладке **Editor** окна **Options**).

Редактор Visual Basic готов показать вам не только перечень всех свойств и методов, но и все параметры, которые принимает данный метод. Это свойство также работает автоматически: достаточно после имени метода напечатать пробел. Для того чтобы явно вызвать список всех параметров, можно воспользоваться меню **Edit | Parameter Info** или клавишами <Ctrl>+<Shift>+<I>. Включить или отключить автоматический показ информации о параметрах можно при помощи флажка **Auto Quick Info** на той же вкладке **Editor** окна **Options**.

Список констант (допустимых значений для данного свойства) также появляется автоматически после того, как вы напечатаете знак равенства '='. Можно воспользоваться также меню **Edit | List Constants** или комбинацией клавиш <Ctrl>+<Shift>+<J>. Про сами константы будет рассказано в *гл. 3*.

Ключевые слова VBA и имена доступных в данный момент классов очень удобно вводить при помощи автоматического дополнения слов. Для этого достаточно выбрать меню **Edit | Complete Word** или нажать клавиши <Ctrl>+<Пробел>. Можно предварительно ничего не печатать, а можно набрать одну-две буквы.

Приведем еще несколько моментов, связанных с редактором кода:

- если вы напечатаете одну строку кода с отступом, то такой же отступ будет установлен для следующих строк. Изменить поведение можно при помощи параметра **Auto Indent** в диалоговом окне **Options**;
- если редактор кода распознает ключевое слово, он автоматически делает его первую букву заглавной и выделяет все слово синим цветом;
- часто бывает необходимо закомментировать или раскомментировать несколько строк сразу. Для этой цели можно включить отображение панели инструментов **Edit** и воспользоваться кнопками **Comment Block** и **Uncomment Block**;
- если при создании процедуры вы пишете ключевое слово `Sub` или `Function`, то редактор автоматически дописывает оператор `End Sub` или `End Function`. Между процедурами вставляется строка-разделитель;
- если при переходе на новую строку редактор кода обнаружит синтаксическую ошибку, то вам будет выдано предупреждение. Меня, например, такое поведение обычно сильно раздражает. Отменить протесты редактора можно, сняв флажок **Auto Syntax Check** в диалоговом окне **Options**. Работе это сильно не повредит, потому что синтаксически неверные строки в любом случае будут автоматически выделяться красным цветом;
- в редакторе кода вполне допускается работа сразу с несколькими окнами редактирования кода. Переход между ними осуществляется по клавишам <Ctrl>+<Tab> или <Ctrl>+<F6>;
- по умолчанию редактор кода работает в режиме **Full Module View** — показ всего содержимого модуля. Если вы хотите просматривать процедуры по отдельности, переключитесь в режим **Procedure View**. Кнопки для переключения находятся в левом нижнем углу окна редактора кода.

2.4. Работа со справкой

Работа со справкой по программированию в Office не так очевидна, как может показаться на первый взгляд.

Вызов окна справки производится из редактора Visual Basic по нажатию клавиши <F1>. Второй вариант — воспользоваться кнопкой **Справка** на панели инструментов **Standard**. В результате откроется дополнительное окно, аналогичное представленному на рис. 2.2.

Еще одна возможность вызвать справку — установить указатель мыши в нужное место в окне редактора кода (например, на имя вызываемого метода или используемого свойства) и нажать клавишу <F1>. Преимуществом такого

подхода является то, что при наличии нескольких вариантов (например, объект `Range` и свойство `Range`) вам автоматически откроется нужная страница.

Справка по программированию в приложении Microsoft Office обычно состоит из трех частей:

- первая часть (**Microsoft Excel Visual Basic Reference**, **Microsoft Word Visual Basic Reference** и т. п.) — это справка по объектной модели самого приложения Office;
- вторая часть (**Microsoft Visual Basic Documentation**, она одинакова во всех приложениях Office) — это справка по синтаксису и встроенным функциям самого языка VBA;
- третья часть (**Microsoft Office Visual Basic Reference**, она также одинакова во всех приложениях Office) — это справка по общим возможностям приложений Office: программная работа с панелями инструментов и меню, работа с помощником, организация взаимодействия с Windows SharePoint Services и т. п.

В некоторых приложениях (например, в Microsoft Access) в справку добавлены дополнительные разделы (рис. 2.3) — по объектной модели ADO, по языку SQL и т. п.

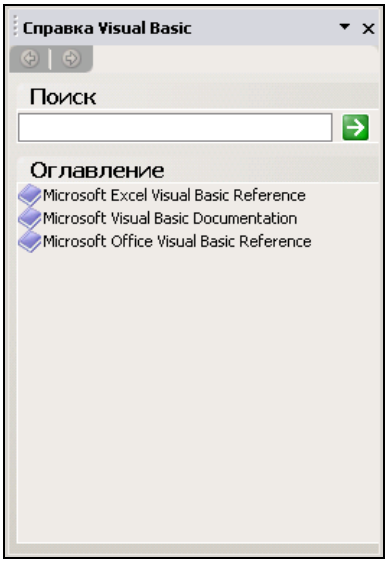


Рис. 2.2. Справка VBA в Excel

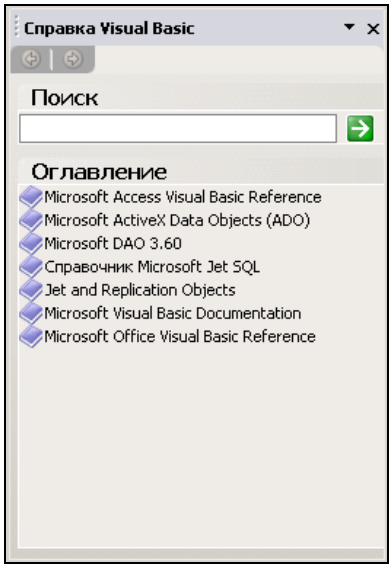


Рис. 2.3. Справка VBA в Access

Обычно самый важный раздел справки — это раздел, который посвящен возможностям конкретного приложения Office. Этот раздел условно можно разделить на две главные части (рис. 2.4):

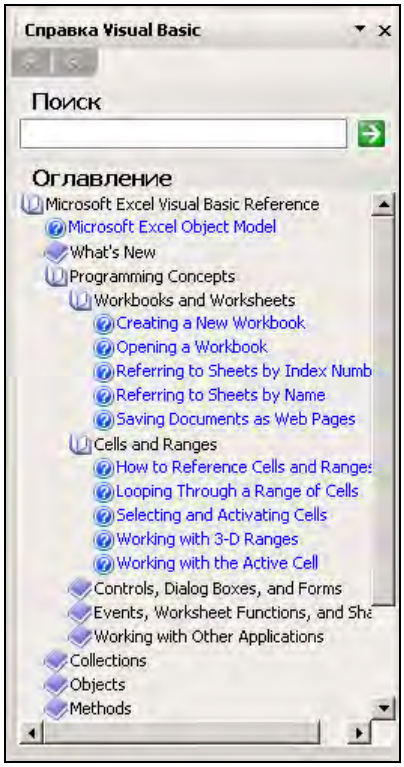


Рис. 2.4. Справка по компонентам объектной модели Excel

- *Programming Concepts (Концепции программирования)* — в этой части рассказывается, как программным образом выполнять самые распространенные операции. Например, для Excel это возможность создать или открыть рабочую книгу, найти нужный лист, получить или записать информацию в ячейку и т. п.;
- *справка по компонентам объектной модели приложения Office*: коллекциям (**Collections**), объектам (**Objects**), методам (**Methods**), свойствам (**Properties**) и т. п. При этом самые важные моменты, которые относятся скорее к области концепций (какими способами, например, можно создать объект `Range` в Excel), приводятся в справке по соответствующему объекту. Представление о всех функциональных возможностях данного объекта можно получить, только просмотрев подряд все его свойства и методы.

Найти направление, т. е. объект и его свойства и методы, которые нужно использовать в вашей ситуации, можно тремя способами:

- просмотреть раздел **Programming Concepts** в справке — не описана ли там ваша ситуация;

- просматривать все подряд объекты, свойства и методы в справке, пытаясь догадаться, что вам может помочь. Это самый неэффективный способ, поскольку объектов в любом приложении Office сотни (часто используемых — намного меньше, и все они рассмотрены в этой книге). Однако если вам предстоит в течение долгого времени заниматься программированием в каком-либо приложении Office, то имеет смысл потратить несколько дней, чтобы подряд прочитать справку по всем объектам, конспектируя самые важные моменты. Я могу гарантировать, что вы узнаете множество таких возможностей, о которых раньше и не подозревали;
- наиболее разумный способ — выполнить нужные вам операции в макрорекордере, а потом проанализировать созданный им код. Однако гарантировать то, что макрорекордер покажет вам самый эффективный путь, невозможно.

И напомним еще один момент, про которые мы уже говорили: справки по VBA на русском языке, к сожалению, не существует. Возможно, в какой-то степени ее сможет заменить эта книга, в которой рассмотрено множество свойств и методов самых важных объектов приложений Office.

Задание для самостоятельной работы 2: Редактирование макроса

ЗАДАНИЕ:

Измените созданный вами в задании для самостоятельной работы к *гл. 1* макрос таким образом, чтобы он запрашивал фамилию ответственного исполнителя. Для этого:

- добавьте вначале кода макроса (над первой строкой `Selection.TypeText`) следующие строки:

```
Dim sInput As String
sInPut = InputBox("Введите фамилию ответственного исполнителя",
"Запрос данных")
```

- замените строку:

```
Selection.TypeText Text:=("Отв. исполнитель Петрова М. М.")
```

(фамилия должна быть ваша) на строку:

```
Selection.TypeText Text:=("Отв. исполнитель " & sInPut)
```

Сохраните измененный макрос, закройте окно редактора кода и убедитесь, что макрос теперь работает по-новому.

Примечание

Не волнуйтесь, что вводимый вами код может показаться непонятным. Почему он именно такой, станет ясно после рассмотрения встроенных функций Visual Basic (функция `InputBox()`) и объектной модели Word (метод `TypeText()` объекта `Selection`) в соответствующих главах. Задача этой самостоятельной работы — позволить вам освоиться в окне редактора кода.

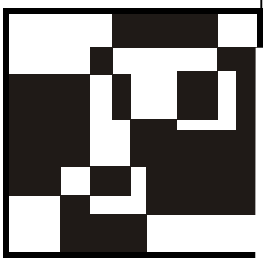
Ответ к заданию 2

1. Откройте Word и нажмите клавиши <Alt>+<F11>. В открывшемся окне Microsoft Visual Basic найдите окно **Project Explorer**, раскройте в нем узел **Normal | Modules | NewMacros**, выделите **NewMacros** и нажмите клавишу <F7>. Откроется окно с кодом **NewMacros**.
2. Найдите вашу процедуру `Sub Подпись()` и внесите необходимые изменения. Общий текст процедуры может быть, например, таким (обратите внимание: код вполне может не совпадать с вашим текстом — все зависит о того, какие действия вы выполняли в макрорекордере):

```
Sub Подпись()
'
' Подпись Макрос
' Макрос записан 02.05.2004 R
'
    Dim sInPut As String
    sInPut = InputBox("Введите фамилию ответственного исполнителя", _
                      "Запрос данных")
    Selection.TypeText Text:="Генеральный директор:" & vbTab & vbTab _
                      & vbTab & "Иванов А. А."
    Selection.ParagraphFormat.Alignment = wdAlignParagraphCenter
    Selection.TypeParagraph
    Selection.ParagraphFormat.Alignment = wdAlignParagraphLeft
    Selection.TypeText Text:=("Отв. исполнитель " & sInPut)
    Selection.TypeParagraph
    Selection.TypeText Text:="т. 55-55"
End Sub
```

3. Нажмите клавиши <Ctrl>+<S>, чтобы сохранить изменения, и <Alt>+<Q>, чтобы вернуться в Word. Выполните макрос, чтобы убедиться, что он работает в соответствии с заданием.

ГЛАВА 3



Синтаксис и программные конструкции VBA

3.1. Основы синтаксиса

Мы подошли к теме, которая многим пользователям покажется самой скучной, — синтаксис языка VBA. Относиться к этой теме, как мне кажется, следует так же, как к изучению азбуки или таблицы умножения: в самой азбуке или таблице умножения ничего интересного нет, но без их знания не удастся читать интересные книги или производить важные вычисления. Кроме того, VBA изначально проектировался и создавался как язык программирования, максимально дружелюбный по отношению к пользователю, который не является профессиональным программистом.

Для тех, кто хорошо знаком с обычным Visual Basic, в этой главе не будет почти ничего нового. Те, кто обладают опытом работы с любым другим современным языком программирования (C++, Java, Delphi, VBScript и JavaScript, Perl и т. п.) также освоят изложенный далее материал почти мгновенно. А тем, кто никогда не сталкивался ни с одним языком программирования, стоит просто выучить то, что изложено в этой главе, и постараться в течение какого-то времени активно применять полученные знания на практике, чтобы они не успели забыться. Потраченные усилия окупятся многократно, тем более что материала на самом деле не так и много — язык VBA очень прост.

Теперь рассмотрим некоторые общие моменты, связанные с синтаксисом языка VBA.

Синтаксис VBA, как понятно из самого названия этого языка (которое расшифровывается как Visual Basic for Applications), почти полностью совпадает с синтаксисом Visual Basic.

Основные синтаксические принципы этого языка следующие:

- VBA нечувствителен к регистру;
- чтобы закомментировать код до конца строки, используется одинарная кавычка (') или команда REM;
- символьные значения должны заключаться в двойные кавычки (");
- максимальная длина любого имени в VBA (переменные, константы, процедуры) — 255 символов;
- начало нового оператора — перевод на новую строку (точка с запятой, как в C, Java, JavaScript, для этого не используется);
- ограничений на максимальную длину строки нет (хотя в редакторе в строке помещается только 308 символов). Несколько операторов в одной строке разделяются двоеточиями:

```
MsgBox "Проверка 1" : MsgBox "Проверка 2"
```

- для удобства чтения можно объединить несколько физических строк в одну логическую при помощи пробела и знака подчеркивания после него:

```
MsgBox "Сообщение пользователю" _
    & vUserName
```

3.2. Операторы

Оператор — это наименьшая способная выполняться единица кода VBA. Оператор может объявлять или определять переменную, устанавливать параметр компилятора VBA или выполнять какое-либо действие в программе.

Арифметических операторов в VBA всего 7. Четыре стандартных: сложение (+), вычитание (–), умножение (*), деление (/), и еще три:

- возведение в степень (^). Например, `2^3 = 8`;
- целочисленное деление (\). Делит первое число на второе, отбрасывая (не округляя) дробную часть. Например, `5\2 = 2`;
- деление по модулю (Mod). Делит первое число на второе, возвращая только остаток от деления. Например, `5 Mod 2 = 1`.

Оператор присваивания в VBA — это знак равенства. Можно записывать так:

```
Let nVar = 10
```

а можно еще проще:

```
nVar = 10
```

Здесь не путайте знак равенства с оператором равенства. Последнее выражение означает "присвоить переменной `nVar` значение 10", а если строка выглядит так:

```
If (nVar = 10)
```

то это значит "если значение переменной `nVar` равно 10".

Если переменной нужно назначить объект, то делается это другими способами.

Операторов сравнения в VBA всего 8:

- равенство (=). Например, `If (nVar = 10)`;
- больше, чем (>) и меньше, чем (<). Например, `If (nVar > 10)`;
- больше или равно (>=) и меньше или равно (<=). Например, `If (nVar >= 10)`;
- не равно (<>). Например, `If (nVar<>10)`;
- сравнение объектов (`Is`). Определяет, ссылаются объектные переменные на один и тот же объект или на разные. Например, `If (obj1 is obj2)`;
- подобие (`Like`). Сравнивает строковый объект с шаблоном и определяет, подходит ли шаблон.

Операторы сравнения всегда возвращают `True` (если утверждение истинно) или `False` (если утверждение ложно).

Приведем некоторые особенности сравнения строковых значений:

- при сравнении строковых значений учитывается регистр;
- пробелы в строковых значениях также учитываются;
- при сравнении текстовых строк на больше/меньше по умолчанию сравниваются просто двоичные коды символов — какие больше или меньше. Если нужно использовать тот порядок, который идет в алфавите, то нужно воспользоваться командой

  ```
  Option Compare Text
  ```

Общий синтаксис оператора `Like` выглядит так:

Выражение1 `Like` *Выражение2*

При этом *Выражение1* — это любое текстовое выражение VBA, а *Выражение2* — шаблон, который передается оператору `Like`. В этом шаблоне можно использовать специальные подстановочные символы (табл. 3.1).

Очень часто при проверке нескольких условий используются логические операторы:

- `And` — логическое И. Должны быть истинными оба условия;
- `Or` — логическое ИЛИ. Должно быть истинным хотя бы одно из условий;

- `Not` — логическое отрицание. Возвращает `True`, если условие ложно;
- `Xor` — логическое исключение. В выражении `E1 Xor E2` возвращает `True`, если только `E1 = True` или только `E2 = True`, иначе — `False`;
- `Eqv` — эквивалентность двух выражений, возвращает `True`, если они имеют одинаковое значение;
- `Imp` — импликация, `E1 Imp E2` возвращает `False`, если `E1 = True` и `E2 = False`, иначе — `True`.

Помнить нужно про `And`, `Or`, `Not`, остальные логические операторы используются редко.

Таблица 3.1. *Подстановочные символы для оператора* `Like`

Подстановочный символ	Значение
#	Любая одна цифра от 0 до 9
*	Любое количество любых символов (включая нулевое)
?	Любой один символ
[a,b,c]	Любой один символ из приведенного в квадратных скобках списка
[!a,b,c]	Любой один символ, кроме приведенных в списке

Почти в любой программе VBA используются операторы конкатенации, т. е. слияния строковых значений. В VBA их два — (+) или (&). Рекомендуется всегда использовать оператор (&), потому что:

- при использовании (&) производится автоматическое преобразование числовых значений в строковые — нет опасности допустить ошибку;
- при использовании оператора (+) сложение строкового значения со значением типа `Null` дает `Null`.

Пример использования оператора (&):

```
MsgBox "Сообщение пользователю " & vUserName
```

Порядок применения операторов выглядит так: вначале в выражении вычисляются арифметические операторы, затем операторы конкатенации, следующими идут операторы сравнения и уже в самом конце логические. Если в выражении есть несколько операторов одного типа, то они применяются в обычном порядке — слева направо. При необходимости можно изменять порядок применения операторов при помощи круглых скобок.

3.3. Переменные и типы данных

Переменные — это контейнеры для хранения изменяемых данных. Без них не обходится практически ни одна программа. Для простоты переменную можно сравнить с номерком в гардеробе — вы сдаете в гардероб какие-то данные, в ответ вам выдается номерок. Когда вам опять потребовались эти данные, вы "предъявляете номерок" и получаете их. Пример работы с переменными в VBA может выглядеть так:

```
Dim nMyAge As Integer
nMyAge = nMyAge + 10
MsgBox nMyAge
```

Перед работой с переменной настоятельно рекомендуется ее объявить. Объявление переменной в нашем примере выглядит так:

```
Dim nMyAge As Integer
```

Расшифруем эту строку.

`Dim` — это область видимости переменной. В VBA предусмотрено 4 ключевых слова для определения области видимости переменных.

- `Dim` — используется в большинстве случаев. Если переменная объявлена как `Dim` в области объявлений модуля, то она будет доступна во всем модуле, если в процедуре — только на время работы этой процедуры.
- `Private` — при объявлении переменных в стандартных модулях VBA значит то же, что и `Dim`. Отличия проявляются только при создании своих классов (эта тема в данной книге не рассматривается).
- `Public` — такая переменная будет доступна всем процедурам во всех модулях данного проекта, если вы объявили ее в области объявлений модуля. Если вы объявили ее внутри процедуры, она будет вести себя как `Dim`.
- `Static` — такие переменные можно использовать только внутри процедуры. Эти переменные видны только внутри процедуры, в которой они объявлены, зато они сохраняют свое значение между разными вызовами этой процедуры. Обычно используются для накопления каких-либо значений. Например:

  ```
  Static nVar1 As Integer
  nVar1 = nVar1 + 1
  MsgBox nVar1
  ```

Если нет никаких особых требований, то имеет смысл всегда выбирать область видимости `Dim`.

Второе слово в нашем объявление (`nMyAge`) — это идентификатор (проще говоря, имя) переменной. Правила выбора имен в VBA едины для многих элементов (переменные, константы, функции, процедуры и т. п.):

- имя должно начинаться с буквы;
- не должно содержать пробелов и символов пунктуации (исключение — символ подчеркивания);
- максимальная длина — 255 символов;
- должно быть уникальным в текущей области видимости (подробнее *см. в разд. 3.3*);
- зарезервированные слова (те, которые подсвечиваются синим цветом в окне редактора кода) использовать нельзя.

При создании программ VBA настоятельно рекомендуется определиться с правилами, по которым будут присваиваться имена объектам — соглашение об именовании. Чаще всего используется так называемое венгерское соглашение (в честь одного из программистов Microsoft, Charles Simonyi, венгра по национальности):

- имя переменной должно начинаться с префикса, записанного строчными буквами. Префикс указывает, что именно будет храниться в этой переменной:
  - `str` (или `s`) — `String`, символьное значение;
  - `fn` (или `f`) — функция;
  - `sub` — процедура;
  - `c` (или все буквы имени заглавные) — константа;
  - `b` — `Boolean`, логическое значение (`True` или `False`);
  - `d` — дата;
  - `obj` (или `o`) — ссылка на объект;
  - `n` — числовое значение;
- имена функций, методов и каждое слово в составном слове должно начинаться с заглавной буквы:

```
MsgBox objMyDocument.Name
Sub CheckDateSub()
```

- в ранних версиях VB не было слова `Const`, все константы определялись как переменные, а для отличия их записывали заглавными буквами, между словами ставили символ подчеркивания, например `COMPANY_NAME`.

Многие программисты используют такой подход для обозначения констант и сейчас (но использование ключевого слова `Const` теперь обязательно — об этом будет рассказано в *разд. 3.4*).

Третья часть нашего объявления (`As Integer`) — это указание на тип данных нашей переменной. Тип данных определяет, данные какого вида можно будет хранить в этой переменной.

В VBA предусмотрены следующие типы данных:

- *числовые*:
  - `Byte` — целое число от 0 до 255;
  - `Integer` — целое число от –32 768 до 32 767;
  - `Long` — большое целое число от –2 147 483 648 до 2 147 483 647;
  - `Currency` — большое десятичное число с 19 позициями, включая 4 позиции после запятой;
  - `Decimal` — еще большее десятичное число с 29 позициями (после запятой можно использовать от 0 до 28 позиций);
  - `Single` и `Double` — значения с плавающей запятой (`Double` в 2 раза больше));

Внимание!

Попытка объявить переменную с типом `Decimal` (например, `Dim n As Decimal`) приведет к синтаксической ошибке. Чтобы получить возможность работать с типом `Decimal`, переменную нужно изначально объявить как `Variant` или вообще объявить без типа (`Dim n`), поскольку тип данных `Variant` используется в VBA по умолчанию. Если мы присвоим переменной с типом `Variant` числовое значение, для которого подходит только `Decimal`, VBA автоматически назначит ей этот тип данных. Кроме того, можно явно указать подтип `Decimal` для переменной типа `Variant` при помощи функции `CDec()`.

- *строковые* (`String` переменной длины (примерно до 2 млрд символов) и фиксированной длины (примерно до 65 400 символов));
- *дата и время* (`Date` — от 01.01.100 до 31.12.9999);
- *логический* (`Boolean` — может хранить только значения `True` и `False`);
- *объектный* (`Object` — хранит ссылку на любой объект в памяти);
- `Variant` — специальный тип данных, который может хранить любые другие типы данных.

Можно также использовать пользовательские типы данных, но их вначале нужно определить при помощи выражения `Type`. Обычно пользовательские

типы данных используются как дополнительное средство проверки вводимых пользователем значений (классический пример — почтовый индекс).

Приведем некоторые моменты, связанные с выбором типов данных для переменных:

- общий принцип — выбирайте наименьший тип данных, который может вместить выбранные вами значения. Если есть какие-то сомнения — выбирайте больший тип во избежание возникновения ошибок;
- если есть возможность, лучше не использовать типы данных с плавающей запятой (`Single` и `Double`). Работа с ними производится медленнее, кроме того, могут быть проблемы при сравнениях за счет округлений;
- при определении переменных можно использовать так называемые символы определения типа ((`%`) — `Integer`, (`$`) — `String` и т. п.). Например, в нашем примере можно закомментировать строку:

  ```
  ' Dim nVar1 As Integer
  ```

  а во второй строке написать:

  ```
  nVar1% = nVar1% + 1
  ```

  Такой подход является устаревшим и к использованию не рекомендуется.

При объявлении переменной можно не указывать ее тип. Например, наше объявление может выглядеть так:

```
Dim nVar1
```

В этом случае переменная будет автоматически объявлена с типом `Variant`.

В принципе, в VBA можно работать и без объявления переменных. Например, такой код

```
nVar1 = nVar1 + 1
MsgBox nVar1
```

будет вполне работоспособным. Если мы используем переменную в программе без ее объявления, то будет автоматически создана новая переменная типа `Variant`. Однако объявлять переменные нужно обязательно! И при этом желательно явно указывать нужный тип данных. Потому что:

- сокращается количество ошибок: программа с самого начала откажется принимать в переменную значение неправильного типа (например, строковое вместо числового);
- при работе с объектами подсказка по свойствам и методам действует только тогда, когда мы изначально объявили объектную переменную с нужным типом. Например, в Excel два варианта кода будут работать одинаково:

- первый вариант:

```
Dim oWbk As Workbook
Set oWbk = Workbooks.Add
```

- второй вариант:

```
Set oWbk = Workbooks.Add
```

Но подсказка по свойствам и методам объекта `oWbk` будет работать только в первом случае.

Все опытные разработчики вообще запрещают использовать переменные без явного их объявления. Для этого можно воспользоваться специальной командой компилятора (она помещается только в раздел объявлений модуля):

```
Option Explicit
```

а можно вставлять эту команду во все модули при их создании автоматически, установив в окне **Options** флажок **Require Variable Declarations** (меню **Tools | Options**, вкладка **Editor**).

Проиллюстрировать, зачем они это делают, можно на простом примере:

```
Dim n
n = n + 1
MsgBox п
```

С виду код не должен вызывать никаких проблем и просто выводить в окне сообщения единицу. На самом деле он выведет пустое окно сообщения. Причина спрятана очень коварно: в третьей строке `n` — это вовсе не английская буква N, а русская П. На вид в окне редактора кода отличить их очень сложно. В то же время компилятор VBA, встретив такой код, просто создаст новую переменную с типом данных `Variant`, у которой будет пустое значение. На выявление такой ошибки может потребоваться определенное время. В то же время при установленном `Option Explicit` проблема определяется мгновенно.

Можно объявить несколько переменных в одной строке, например, так:

```
Dim nVar1 As Integer, s1 As String
```

Присвоение значений переменным выглядит так:

```
nVar1 = 30
```

Если нужно увеличить уже существующее значение переменной, то команда может выглядеть так:

```
nVar1 = nVar1 + 1
```

В обоих примерах знак равенства означает не "равно", а "присвоить".

При присвоении значений переменным нужно помнить о следующем:

- строковые значения всегда заключаются в двойные кавычки:

```
sVar1 = "Hello"
```

- значение дата/время заключается в символы "решетка" (#):

```
dVar1 = #05/06/2004#
```

  Обратите внимание, что при присвоении значения даты/времени таким "явным" способом нам приходится использовать принятые в США стандарты: `05` в данном случае — это месяц, `06` — день. А отображение этого значения (например, в окне сообщения) будет зависеть от региональных настроек на компьютере пользователя;

- если нужно передать шестнадцатеричное значение, то перед ним ставятся символы (`&H`):

```
nVar1 = &HFF00
```

  Для передачи восьмеричного значения (что случается намного реже) нужно использовать набор символов (`&O`).

В переменных до присвоения им значений пользователем содержится:

- в переменных всех числовых типов данных — 0;
- в строковых переменных переменной длины — `""` (строка нулевой длины);
- в строковых переменных фиксированной длины — строка заданной длины с символами ASCII 0 (эти символы на экран не выводятся);
- в `Variant` — специальное пустое значение `Empty`. Произвести проверку на это значение (т. е. было ли присвоено значение переменной или нет) можно при помощи функции `IsEmpty()`;
- в `Object` — ничего (нет ссылки ни на один из объектов).

Задание для самостоятельной работы 3.1: Работа с переменными и операторами

Подготовка:

1. Создайте новую книгу Excel и сохраните ее как C:\LabVariablesOperators.xls. Введите в ячейки A1, A2 и A3 этой книги любые значения.
2. Откройте редактор Visual Basic в Excel и создайте в этой книге новый стандартный модуль *(см. разд. 2.2)*.

3. При помощи меню **Tools | References** добавьте в ваш проект ссылку на библиотеку Microsoft Word 11.0 Object Library.
4. Введите в созданном вами стандартном модуле следующий код:

```
Public Sub FromExcelToWord()
    MsgBox Range("A1").Text
    MsgBox Range("A2").Text
    MsgBox Range("A3").Text

    Dim oWord As Word.Application
    Dim oDoc As Word.Document
    Set oWord = CreateObject("Word.Application")
    oWord.Visible = True
    Set oDoc = oWord.Documents.Add()
    oDoc.Activate
    oWord.Selection.TypeText "Вставляемый текст"
End Sub
```

   Этот код должен выводить в окна сообщений значения ячеек A1, A2 и A3, а затем открыть Word и впечатать в начало нового документа строку `"Вставляемый текст"`.
5. Убедитесь, что код работает без ошибок.

ЗАДАНИЕ:

Измените код этой процедуры таким образом, чтобы вместо строки `"Вставляемый текст"` выводились значения ячеек A1, A2 и A3 вместе.

Ответ к заданию 3.1

Итоговый код может выглядеть так:

```
Public Sub FromExcelToWordAnswer()
    Dim sA1, sA2, sA3, sText As String
    sA1 = Range("A1").Text
    sA2 = Range("A2").Text
    sA3 = Range("A3").Text

    sText = sA1 + " " + sA2 + " " + sA3

    Set oWord = CreateObject("Word.Application")
    oWord.Visible = True
    Set oDoc = oWord.Documents.Add()
    oDoc.Activate
    oWord.Selection.TypeText sText
End Sub
```

3.4. Константы

Константы — еще один контейнер для хранения данных, но, в отличие от переменных, они не изменяются в ходе выполнения VBA-программы. Константы используют в следующих случаях:

- код становится более читаемым, убираются потенциальные ошибки;
- чтобы изменить какое-либо значение в коде (например, уровень налога), это нужно сделать всего один раз — в объявлении константы.

В VBA константы определяются при помощи ключевого слова `Const`:

```
Const COMP_NAME As String = "Microsoft"
```

Главное отличие констант от переменных заключается в том, что при попытке изменить значение константы в теле процедуры будет выдано сообщение об ошибке.

Константы очень удобны при работе с группами именованных элементов (дни недели, месяцы, цвета, клавиши, типы окон и т. п.). Они позволяют использовать в коде программы легко читаемые обозначения вместо труднозапоминаемых числовых кодов. Например, строки:

```
UserForm1.BackColor = vbGreen
```

и

```
UserForm1.BackColor = 65280
```

функционально одинаковы, но в чем смысл первой строки, догадаться гораздо легче.

В VBA встроено множество служебных констант: календарных, для работы с файлами, цветами, формами, типами дисков и т. п. Просмотреть их можно через справочную систему VBA: **Microsoft Visual Basic Documentation | Visual Basic Language Reference | Constants**. Про одну из констант (она находится в разделе **Miscellaneous Constants**) следует сказать особо: константа `vbCrLf` позволяет произвести переход на новую строку, например:

```
MsgBox "Первая строка" + vbCrLf + "Вторая строка"
```

Множество наборов констант встроено в объектные модели приложений Office, которые мы будем рассматривать в *гл. 10—15*.

3.5. Операторы условного и безусловного перехода

Операторы условного перехода — одни из самых важных и часто используемых элементов в языках программирования. Общий принцип их работы прост: проверяется соответствие каким-то условиям (истинность или лож-

ность каких-либо выражений) и в зависимости от этого выполнение программы направляется по одной или другой ветви. В VBA предусмотрено два оператора условного перехода: `If...Then` и `Select Case`.

3.5.1. Оператор *If ... Then*

Оператор `If...Then` — самый популярный у программистов. Полный его синтаксис выглядит так (необязательные части заключены в квадратные скобки):

```
If Условие Then
    Команды1
[ElseIf УсловиеN Then
    КомандыN]
[Else
    Команды2]
End If
```

При этом:

- *Условие* — выражение, которое проверяется на истинность. Если оно истинно, то выполняются *Команды1*, если ложно — *Команды2*;
- *УсловияN* — дополнительные условия, которые также можно проверить. В случае, если они выполняются (выражение *УсловияN* истинно), то выполняются *КомандыN*. Дополнительные условия (вместе с конструкцией `ElseIf`) можно повторять неограниченное количество раз, но если вам нужно реализовать проверку на соответствие большому количеству условий, то правильнее будет использовать конструкцию `Select Case` *(см. разд. 3.5.2)*.

Оператор `If...Then` применяется:

- когда нужно проверить на одно условие и в случае соответствия сделать какое-то действие:

```
If nTemperature < 10 Then
    MsgBox "Надеть куртку"
End If
```

- когда нужно сделать то же, что и в предыдущем примере, а в случае несоответствия выполнить другое действие:

```
If nTemperature < 10 Then
    MsgBox "Надеть куртку"
Else
    MsgBox "Надеть ветровку"
End If
```

- когда нужно проверить на соответствие несколько условий (обратите внимание на использование логических операторов):

```
If (nTemperature < 10) And (bRain = True) Then
    MsgBox "Надеть куртку и взять зонтик"
End If
```

- в случае, когда проверка первого условия вернула `False`, нужно проверить на соответствие еще несколько условий (удобно использовать `ElseIf`):

```
If (bIGoInCar = True) Then
    MsgBox "Одеться для машины"
ElseIf nTemperature < 10 Then
    MsgBox "Надеть куртку"
Else
    MsgBox "Можно идти в рубашке"
End If
```

  В этом примере, поскольку `bIGoInCar` — переменная типа `Boolean` и сама по себе принимает значения `True` или `False`, первая строка может выглядеть так:

```
If bIGoInCar Then ...
```

Приведу некоторые замечания по использованию `If...Then`:

- ключевое слово `Then` должно находиться в одной строке с `If` и условием. Если вы перенесете его на следующую строку, будет выдано сообщение об ошибке;
- если разместить команду, которую нужно выполнить при истинности проверяемого условия, на одной строке с `If` и `Then`, то `End If` можно не писать:

```
If nTemperature < 10 Then MsgBox "Надеть куртку"
```

  Если же вы используете несколько команд или конструкции `Else/ElseIf`, то `End If` в конце нужно писать обязательно, иначе возникнет синтаксическая ошибка;
- для выражения `If...Then` настоятельно рекомендуется использовать отступы для выделения блоков команд. Иначе читать код будет трудно;
- операторы `If...Then` можно вкладывать друг в друга:

```
If MyVar = 5 Then
    MsgBox "MyVar = 5"
    If MyVar = 10 Then
        MsgBox "MyVar = 10"
    End If
End If
```

3.5.2. Оператор *Select Case*

Оператор `Select Case` идеально подходит для проверки одного и того же значения, которое нужно много раз сравнить с разными выражениями. Синтаксис его очень прост:

```
Select Case Выражение
    Case Условие1
        Команды1
    [Case УсловиеN
        КомандыN]
    [Case Else
        Команды2]
End Select
```

Например:

```
Select Case sDayOfWeek
    Case "Понедельник"
        MsgBox "Салат из шпината"
    Case "Вторник"
        MsgBox "Салат из морской капусты"
    Case Else
        MsgBox "На этот день у нас ничего не предусмотрено"
End Select
```

Приведу некоторые замечания по поводу `Select Case`:

- строка:

  ```
  Case "Понедельник"
  ```

  на самом деле означает:

  ```
  Case sDayOfWeek = "Понедельник"
  ```

  Такое равенство подразумевается по умолчанию. Но вы можете использовать и другой оператор сравнения или целый набор операторов, например:

  ```
  Case 0 To 5, 15, Is > 55
  ```

  Такое выражение можно перевести как "Если проверяемое выражение попало в диапазон от 0 до 5 включительно, или равно 15, или больше 55".

  Слово `Is` при этом можно пропустить — компилятор VBA добавит это ключевое слово за вас. Несколько критериев в `Case` перечисляются через запятые и объединяются так, как работает оператор `Or`, т. е. выполнение пойдет по этой ветви, если тестируемое значение будет удовлетворять хотя бы одному из критериев;

- при использовании диапазона (`0 To 5`) включаются и границы диапазона (в данном случае `0` и `5`).

3.5.3. Оператор *GoTo*

Оператор `GoTo` — это оператор безусловного перехода, когда ход выполнения программы без проверки каких-либо условий перепрыгивает на метку в коде. Пример использования `GoTo` может выглядеть так:

```
GoTo EngineNotStarted
    ...
EngineNotStarted:
    MsgBox "Едем на метро"
    ...
```

Здесь `EngineNotStarted:` — это метка, для нее используется имя (выбираемое по правилам назначения имен для переменных), которое оканчивается двоеточием. Эта метка может находиться как до, так и после оператора `GoTo`. В любом случае, при выполнении оператора `GoTo` ход выполнения "перепрыгнет" на указанную в `GoTo` метку.

Иногда использование `GoTo` очень удобно, например, когда нам нужно добиваться от пользователя ввода правильного значения неизвестное число раз. Однако использовать `GoTo` категорически не рекомендуется, потому что код становится трудночитаемым. Чаще всего `GoTo` можно заменить на конструкцию `Do While...Loop` *(см. разд. 3.6)* или на вызов функции из самой себя.

Задание для самостоятельной работы 3.2: Работа с операторами условного перехода

Подготовка:

1. Создайте новую книгу Excel и сохраните ее как C:\LabCondConstructions.xls.
2. Откройте редактор Visual Basic в Excel и создайте в этой книге новый стандартный модуль.
3. Введите в этом модуле следующий код:

```
Public Sub IfThenSub()
    Dim nResult As Integer
    nResult = MsgBox("Нажмите кнопку", vbYesNo, "Окно сообщения")
    ThisWorkbook.Worksheets(1).Range("A1").Value = _
        "Вы нажали кнопку: " & nResult
    ThisWorkbook.Worksheets(1).Range("A1").Columns.AutoFit
End Sub
```

4. Запустите этот код на выполнение и убедитесь, что он выполняется без ошибок. Этот код должен вставлять в ячейку A1 первого листа вашей книги текстовое значение вида "Вы нажали кнопку: 6", в зависимости от того, какая кнопка была нажата в окне сообщения.

Примечание

Получить информацию о том, какие значения при нажатии какой кнопки возвращаются из окна сообщения, можно при помощи справки по функции `MsgBox()`.

ЗАДАНИЕ 3.2, А:

Измените код этой процедуры таким образом, чтобы вместо чисел в ячейку вписывалось строковое значение нажатой кнопки (например, "Вы нажали кнопку: Да"). Используйте при этом синтаксическую конструкцию `If...Then...Else`.

Ответ к заданию 3.2, А:

Итоговый код для вашей процедуры может быть таким:

```
Public Sub IfThenSubAnswer()
    Dim nResult As Integer
    nResult = MsgBox("Нажмите кнопку", vbYesNo, "Окно сообщения")
    If nResult = vbYes Then
        sResult = "Да"
    ElseIf nResult = vbNo Then
        sResult = "Нет"
    Else
        sResult = "Неизвестная кнопка"
    End If
    ThisWorkbook.Worksheets(1).Range("A1").Value = "Вы нажали кнопку: " _
        & sResult
    ThisWorkbook.Worksheets(1).Range("A1").Columns.AutoFit
End Sub
```

ЗАДАНИЕ 3.2, Б:

Замените в вашей процедуре строку:

```
nResult = MsgBox("Нажмите кнопку", vbYesNo, "Окно сообщения")
```

на:

```
nResult = MsgBox("Нажмите кнопку", vbAbortRetryIgnore, "Окно сообщения")
```

Измените вашу процедуру таким образом, чтобы она вставляла в ячейку A1 значения "Прервать", "Повторить" или "Пропустить", в зависимости от того, какая кнопка в окне сообщения была нажата. Используйте при этом синтаксическую конструкцию `Select Case`.

Ответ к заданию 3.2, Б:

Итоговый код для вашей процедуры может быть таким:

```
Private Sub SelectCaseAnswer()
    nResult = MsgBox("Нажмите кнопку", vbAbortRetryIgnore, "Окно
сообщения")
    Select Case nResult
        Case vbAbort
            sResult = "Отменить"
        Case vbRetry
            sResult = "Повторить"
        Case vbIgnore
            sResult = "Пропустить"
        Case Else
            sResult = "Неизвестная кнопка"
    End Select
    ThisWorkbook.Worksheets(1).Range("A1").Value = "Вы нажали кнопку: " _
        & sResult
    ThisWorkbook.Worksheets(1).Range("A1").Columns.AutoFit
End Sub
```

3.6. Работа с циклами

Циклы используются в ситуациях, когда нам нужно выполнить какое-либо действие несколько раз. Первая ситуация — мы знаем, сколько раз нужно выполнить определенное действие, в этом случае используется конструкция `For...Next`:

```
For iCounter = 1 to 10
    MsgBox "Счетчик: " & iCounter
Next
```

Чтобы указать, насколько должно прирастать значение счетчика, используется ключевое слово `Step`:

```
For iCounter = 1 to 10 Step 2
    MsgBox "Счетчик: " & iCounter
Next
```

Можно и уменьшать исходное значение счетчика:

```
For iCounter = 10 to 1 Step -2
    MsgBox "Счетчик: " & iCounter
Next
```

Для безусловного выхода из конструкции `For...Next` используется команда `Exit For`:

```
VStop = InputBox("Введите значение останова")
VInput = CInt(VStop)
For iCounter = 1 to 10
    MsgBox "Счетчик: " & iCounter
    If iCounter =VInput Then Exit For
Next
```

Очень часто в VBA требуется сделать какое-нибудь действие со всеми элементами коллекции или массива — перебрать все открытые документы, все листы Excel, все ячейки в определенном диапазоне и т. п. Для того чтобы пройти циклом по всем элементам коллекции, используется команда `For Each...Next`:

```
For Each oWbk in Workbooks
    MsgBox oWbk.Name
Next
```

При использовании этого приема можно очень просто найти и получить ссылку на нужный нам объект:

```
For Each oWbk in Workbooks
    If oWbk.Name = "Сводка.xls" Then
        Set oMyWorkBook = oWbk
        Exit For
    End If
Next
```

В этом случае мы проходим циклом по всем элементам коллекции `Workbooks` (т. е. по открытым рабочим книгам в Excel), для каждой книги проверяем ее имя и, если нашли книгу с именем `"Сводка.xls"`, получаем на нее ссылку и выходим из цикла. Коллекция рабочих книг — это специальная коллекция, которая умеет производить поиск в себе по имени элемента, поэтому, в принципе, можно было обойтись такой строкой:

```
Set oMyWorkBook = Workbooks("Сводка.xls")
```

Но для многих других коллекций без конструкции `For Each...Next` не обойтись.

Еще одна ситуация: когда мы не знаем точно, сколько раз должна быть выполнена та или другая команда — это зависит от какого-либо условия. В этом случае используются конструкции `Do While...Loop` и `Do Until...Loop`.

Конструкция `Do While...Loop` означает: выполнять какое-либо действие до тех пор, пока условие истинно:

```
Do While MyVar < 10
    MyVar = MyVar + 1
    MsgBox "MyVar = " & MyVar
Loop
```

Применений на практике — множество: пройти по всему набору записей, пока они не закончатся, требовать от пользователя ввести подходящее значение, пока он наконец не введет его, и т. п.

Внимание!

Если вы случайно запустили в своей программе бесконечный цикл, нажмите на клавиши <Ctrl>+<Break>. Откроется окно, аналогичное представленному на рис. 3.1, в котором вы сможете продолжить выполнение (кнопка **Continue**), завершить его (**End**) или открыть ваш код в отладчике (**Debug**).

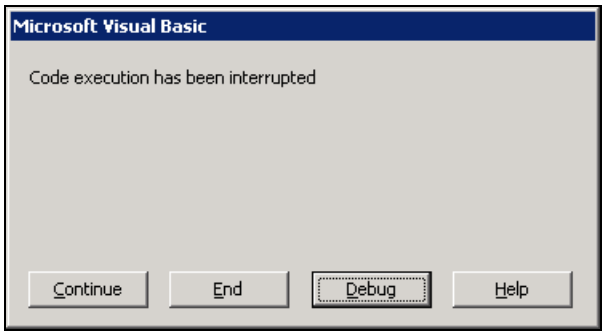


Рис. 3.1. Выполнение макроса остановлено по <Ctrl>+<Break>

Второй вариант — `Do Until...Loop`. Все выглядит точно так же, за одним исключением: цикл будет выполняться до тех пор, пока условие ложно.

```
Do Until MyVar >= 10
    MyVar = MyVar + 1
    MsgBox "MyVar = " & MyVar
Loop
```

Можно переписать цикл так, чтобы условие проверялось после завершения цикла:

```
Do
    MyVar = MyVar + 1
    WScript.Echo "MyVar = " & MyVar
Loop While MyVar < 10
```

В этом случае цикл будет выполнен, по крайней мере, один раз.

Немедленный выход из цикла можно произвести по команде `Exit Do`.

В VBA имеется также конструкция `While...Wend`. Это еще один вариант цикла, который оставлен для обратной совместимости с первыми версиями Visual Basic. Функциональные возможности — те же, что и у конструкции `Do While...Loop`:

```
While My Var < 10
    MyVar = MyVar + 1
    WScript.Echo "MyVar = " & MyVar
Wend
```

3.7. Массивы

Массивы используются для хранения в памяти множества значений. Вместо того чтобы объявлять множество похожих друг на друга переменных, часто гораздо удобнее воспользоваться массивом.

Объявление массива производится очень просто:

```
Dim MyArray(2) As Integer
```

Здесь `2` — это верхняя граница массива *(upper bound)*. По умолчанию нижней границе массива *(lower bound)* соответствует элемент с номером 0. Количество элементов, которое может хранить массив, — от 0 до верхней границы включительно. Наш массив может хранить три целочисленных элемента.

Если вам хочется, чтобы нижняя граница массива (и, соответственно, нумерация элементов) начиналась с 1, то в раздел объявлений модуля нужно внести команду:

```
Option Base 1
```

В принципе, тип данных для массива можно не объявлять:

```
Dim MyArray(2)
```

В этом случае для элементов массива будет использован тип `Variant`. Такой массив сможет хранить в себе элементы разных типов данных, но требования к памяти у него будут выше и работать он будет чуть медленнее по сравнению с массивом, для которого тип данных указан явно (например, `Integer` или `String`).

Присвоить значение отдельному элементу массива (в нашем случае — первому) можно очень просто:

```
MyArray(0) = 100
```

А затем это значение можно будет извлечь:

```
MsgBox MyArray(0)
```

Массивы могут быть многомерными, в частности, двумерными:

```
Dim MyArray(4, 9)
```

В каждой строке многомерного массива удобно хранить данные, относящиеся к одному объекту (например, имя сотрудника, уникальный номер, номер телефона). В VBScript в одном многомерном массиве может быть до 60 измерений.

Часто необходимо использовать *динамические* массивы, размер которых можно изменять в ходе выполнения. Динамический массив объявляется следующим образом:

```
Dim MyArray()        ' объявляем массив без верхней границы
ReDim MyArray(4)     ' изменяем размер массива
```

Команда `ReDim` не только изменяет размер массива, но и удаляет из него все старые значения. Чтобы старые значения сохранились, используется ключевое слово `Preserve`:

```
ReDim Preserve MyArray(7)
```

Однако если новый размер массива меньше, чем количество помещенных в него элементов, слово `Preserve` не поможет — часть данных все равно будет потеряна.

Массивы можно создавать и заполнять одновременно:

```
Dim MyArray
MyArray = Array(100, 200, 300, 400, 500)
```

Указывать размер массива необязательно — он будет автоматически настроен в соответствии с количеством передаваемых элементов.

Очистить массив можно командой `Erase`:

```
Erase MyArray
```

Массив фиксированной длины просто очищается, динамический массив разинициализируется — его придется инициализировать (определять размер) заново.

В динамических массивах часто не известно, сколько элементов в массиве. Для определения количества элементов используется функция `UBound()` (если массив одномерный или вас интересует размер первого измерения, то *измерение* передавать не надо):

```
UBound(имя_Массива [, измерение])
```

Как ни удивительно, но при программировании в VBA вам редко придется сталкиваться с массивами. Вместо них в объектных моделях приложений

Office обычно используются коллекции. *Коллекции* — это специальные объекты, которые предназначены для хранения наборов одинаковых элементов. Например, в Word предусмотрена коллекция `Documents` для хранения элементов `Document`, т. е. всех открытых документов, в Excel — коллекции `Workbooks` (открытые книги) и `Worksheets` (листы в книге) и т. п. Коллекции обычно удобнее, чем массивы: они изначально безразмерны и в них предусмотрен стандартный набор свойств и методов (метод `Add()` для добавления нового элемента, свойство `Count` для получения информации о количестве элементов, метод `Item()` для получения ссылки на нужный элемент). Для многих коллекций в объектных моделях кроме стандартных предусмотрен еще и набор специфических свойств и методов.

Задание для самостоятельной работы 3.3: Работа с циклами

Подготовка:

1. Создайте новый документ Word с именем C:\LabLoops.doc. Введите в этом документе несколько предложений с текстом.
2. Откройте редактор Visual Basic и создайте в этом документе новый стандартный модуль.

ЗАДАНИЕ 3.3, А:

Создайте в этом стандартном модуле процедуру `ForNextSub()`, которая выводила бы последовательно 10 окон сообщений с цифрами от 1 от 10.

Ответ к заданию 3.3, А:

Итоговый код процедуры может быть таким:

```
Public Sub ForNextSub()
    Dim i As Integer
    For i = 1 To 10
        MsgBox i
    Next
End Sub
```

или таким:

```
Public Sub ForNextSub2()
    Dim i As Integer
    i = 1
    Do While i <= 10
        MsgBox i
```

```
        i = i + 1
    Loop
End Sub
```

ЗАДАНИЕ 3.3, Б:

Создайте в этом программном модуле процедуру `ForEachSub()`, которая выводила бы в окна сообщений последовательно каждое слово в вашем документе LabLoops.doc.

Коллекцию (массив) всех слов документа можно получить при помощи конструкции `ThisDocument.Words`. Значение каждого слова можно получить при помощи свойства `Text` элемента этой коллекции.

Ответ к заданию 3.3, Б:

Итоговый код процедуры `ForEachSub()` может быть таким:

```
Public Sub ForEachSub()
    For Each oWord In ThisDocument.Words
        MsgBox oWord.Text
    Next
End Sub
```

3.8. Процедуры и функции

3.8.1. Виды процедур

Процедуры — это самые важные функциональные блоки языка VBA. В VBA вы можете выполнить только тот программный код, который содержится в какой-либо процедуре (обычной в стандартном модуле, событийной для элемента управления на форме и т. п.). Иногда начинающие пользователи пытаются записать команды прямо в область объявлений стандартного модуля и не могут понять, почему они не выполняются (сообщения об ошибке при этом не выдается — просто этот код становится "невидим" для компилятора). Причина проста: в разделе объявлений модуля (когда в верхних списках редактора кода показываются значения **General** и **Declarations**) могут быть только объявления переменных уровня модуля и некоторые специальные инструкции для компилятора. Весь остальной программный код должен находиться внутри процедур.

В VBA предусмотрены следующие типы процедур:

- процедура типа `Sub` (подпрограмма) — универсальная процедура для выполнения каких-либо действий:

```
Sub Farewell()
    MsgBox "Goodbye"
End Sub
```

Макрос в VBA — это процедура типа `Sub`, не имеющая параметров. Только макросы можно вызывать по имени из редактора VBA или из приложения Office. Все другие процедуры нужно вызывать либо из других процедур, либо специальными способами, о которых будет рассказано далее;

- процедура типа `Function` (функция) — набор команд, которые должны быть выполнены. Принципиальное отличие только одно: функция возвращает вызвавшей ее программе (или процедуре) какое-то значение, которое будет там использовано. Пример функции:

```
Function Tomorrow()
    Tomorrow = DateAdd("d", 1, Date())
End Function
```

и пример ее вызова:

```
Private Sub Test1()
    Dim dDate
    dDate = Tomorrow()
    MsgBox dDate
End Sub
```

В тексте функции необходимо предусмотреть оператор, который присваивает ей какое-либо значение. В нашем случае это строка:

```
Tomorrow = DateAdd("d", 1, Date())
```

В принципе, процедуры типа `Sub` тоже могут возвращать значения — при помощи переменных, передаваемых по ссылке *(см. разд. 3.8.4)*. Зачем же тогда нужны функции? Все очень просто: функцию можно вставлять практически в любое место программного кода. Например, наш последний пример может выглядеть намного проще:

```
Private Sub Test1()
    MsgBox Tomorrow()
End Sub
```

В VBA предусмотрены сотни встроенных функций (и гораздо большее количество функций предусмотрено в объектных моделях приложений Office). Даже в нашем примере используются две встроенные функции: `Date()`, которая возвращает текущую дату по часам компьютера, и `DateAdd()`, которая умеет прибавлять к текущей дате определенное количество дней, недель, месяцев, лет и т. п. Про встроенные функции будет рассказано в *разд. 3.9*.

В VBA имеются также *процедуры обработки событий* (event procedure) — процедуры типа `Sub` специального назначения, которые выполняется в случае возникновения определенного события, например, при открытии формы или нажатии на ней кнопки. Про события подробнее будет рассказано в *гл. 5*.

Есть еще процедуры типа `Property` (процедуры свойства). Они нужны для определения свойств создаваемого вами класса, а поскольку созданием своих классов мы заниматься не будем, то их можно не рассматривать.

3.8.2. Область видимости процедур

По умолчанию все процедуры VBA (за исключением процедур обработки событий) определяются как *открытые* (`Public`). Это значит, что их можно вызвать из любой части программы — из того же модуля, из другого модуля, из другого проекта. Объявить процедуру как `Public` можно так:

```
Public Sub Farewell()
```

или, поскольку процедура определяется как `Public` по умолчанию, то можно и так:

```
Sub Farewell()
```

Можно объявить процедуру локальной:

```
Private Sub Farewell()
```

В этом случае эту процедуру можно будет вызвать только из того модуля, в котором она расположена. Такое решение иногда может предотвратить ошибки, связанные с вызовом процедур, не предназначенных для этого, из других модулей.

Можно ограничить область видимости открытых процедур (тех, которые у вас определены как `Public`) в каком-то модуле рамками одного проекта. Для этого достаточно в разделе объявлений этого модуля вписать строку

```
Option Private Module
```

Если при объявлении процедуры использовать ключевое слово `Static`, то все переменные в этой процедуре автоматически станут статическими и будут сохранять свои значения и после завершения работы процедуры *(см. разд. 3.3)*. Например:

```
Private Static Sub Farewell()
```

3.8.3. Объявление процедур

Объявить процедуру можно вручную, например, добавив в код строку:

```
Private Sub Farewell()
```

При этом редактор кода автоматически добавит строку `End Sub` и линию-разделитель. А можно объявить процедуру, воспользовавшись меню **Insert | Procedure**. Разницы нет никакой.

3.8.4. Передача параметров

Параметры — это значения, которые передаются от одной процедуры к другой. В принципе, можно обойтись и без параметров, воспользовавшись только переменными уровня модуля, но при использовании параметров читаемость программы улучшается. Чтобы процедура имела возможность принимать параметры, ее вначале нужно объявить с параметрами. Далее приведен пример простой функции, которая складывает два числа и возвращает результат:

```
Function fSum(nItem1 As Integer, nItem2 As Integer)
    fSum = nItem1 + nItem2
End Function
```

Ее вызов может выглядеть так:

```
MsgBox fSum(3, 2)
```

В данном случае мы объявили оба параметра как обязательные, и поэтому попытка вызвать функцию без передачи ей какого-либо параметра (например, `fSum(3)`) приведет к ошибке "Argument not optional" — "Параметр не является необязательным". Чтобы можно было пропускать какие-то параметры, их нужно сделать необязательными. Для этой цели используется ключевое слово `Optional`:

```
Function fSum(nItem1 As Integer, Optional nItem2 As Integer)
```

В справке по встроенным функциям VBA необязательные параметры заключаются в квадратные скобки.

Для проверки того, был ли передан необязательный параметр, используется либо функция `IsMissing` (если для этого параметра был использован тип `Variant`), либо его значение сравнивается со значениями переменных по умолчанию (ноль для числовых данных, пустая строка для строковых и т. п.).

Вызов функции с передачей параметров может выглядеть так:

```
nResult = fSum(3, 2)
```

Однако здесь есть несколько моментов, которые необходимо рассмотреть.

В нашем примере мы передаем параметры по позиции, т. е. значение 3 присваивается первому параметру (`nItem1`), а значение 2 — второму (`nItem2`). Однако параметры можно передавать и по имени:

```
nResult = fSum(nItem1 := 3, nItem2 := 2)
```

Обратите внимание, что, несмотря на то, что здесь выполняется вполне привычная операция — присвоение значений, оператор присваивания используется не совсем обычный — двоеточие со знаком равенства (:=), как в C++. При использовании знака равенства возникнет ошибка.

Конечно, вместо явной передачи значений (`fSum(3, 2)`) можно использовать переменные. Однако что произойдет с переменными после того, как они "побывают" в функции, если функция изменяет их значения? Останутся ли эти значения за пределами функции прежними или изменятся?

Все зависит от того, как именно передаются параметры: *по ссылке* (по умолчанию, можно также использовать ключевое слово `byRef`) или *по значению* (ключевое слово `byVal`).

Если параметры передаются по ссылке, то фактически в вызываемую процедуру передается ссылка на эту переменную в оперативной памяти. Если эту переменную в вызываемой процедуре изменить, то значение изменится и в вызывающей функции. Это принятое в VBA поведение по умолчанию.

Если параметры передаются по значению, то фактически в оперативной памяти создается копия этой переменной и в вызываемую процедуру передается эта копия. Конечно же, чтобы вы не сделали с копией, на исходную переменную это никак не повлияет и в вызывающей процедуре не отразится.

Продемонстрировать разницу можно на простом примере:

```
Private Sub TestProc()
    'Объявляем переменную nPar1 и присваиваем ей значение
    Dim nPar1 As Integer
    nPar1 = 5

    'Передаем ее как параметр nItem1 функции fSum
    MsgBox fSum(nItem1:=nPar1, nItem2:=2)

    'А теперь проверяем, что стало в нашей переменной nPar1
    'после того, как она побывала в функции fSum
    MsgBox nPar1
End Sub

Function fSum(nItem1 As Integer, nItem2 As Integer)
    'Используем значение переменной
    fSum = nItem1 + nItem2
    'А затем ее меняем!
    nItem1 = 10
End Function
```

Проверьте, что будет, если поменять строку объявления функции:

```
Function fSum(nItem1 As Integer, nItem2 As Integer)
```

на другую:

```
Function fSum(byVal nItem1 As Integer, nItem2 As Integer)
```

Можно продемонстрировать компилятору VBA, что значение, возвращаемое функцией, нас совершенно не интересует. Для этого достаточно не заключать ее параметры в круглые скобки. Например, для нашей функции это может выглядеть так:

```
fSum 3, 2
```

Такой код будет работать совершенно нормально. Однако если нам потребуется все-таки узнать, что возвращает функция, то придется передаваемые ей параметры заключать в круглые скобки:

```
nResult = fSum(3, 2)
```

Для многих встроенных функций компилятор VBA в принципе не дает возможности игнорировать возвращаемое значение, заставляя помещать параметры в круглые скобки и принимать возвращаемое значение. В противном случае он сообщает о синтаксической ошибке.

3.8.5. Вызов и завершение работы процедур

С примерами вызова процедур и функций из кода мы уже сталкивались. Для этого достаточно записать имя процедуры или функции, передать ей необходимые параметры, а для функции еще и принять возвращаемое значение. Одни процедуры можно вызывать из других процедур. Но как дать пользователю возможность запустить самую первую процедуру?

В нашем распоряжении следующие возможности:

- создать макрос (т. е. специальную процедуру, не принимающую параметров, в модуле **NewMacros**) и запустить его по имени, кнопке или комбинацией клавиш *(см. разд. 1.4)*. Макрос затем может вызывать и другие процедуры;
- создать форму и воспользоваться набором событий этой формы и элементов управления на ней (об этом будет рассказано в *гл. 5*) или просто элементом управления на листе Excel или документе Word;
- назначить процедуре специальное имя (`AutoExec()`, `AutoNew()` и т. п.). Полный список таких специальных имен можно посмотреть в документации. Правда, поскольку раньше эти возможности активно использовались вирусами, в Office 2003 по умолчанию эти макросы запускаться не будут. Для того чтобы обеспечить им (и многим другим макросам) возможность

запуска, необходимо изменить установленный уровень безопасности в меню **Сервис | Макрос | Безопасность** или обеспечить цифровые подписи для ваших макросов;

- вместо специального имени для макроса использовать событие: например, событие запуска приложения, событие открытия документа и т. п. Это рекомендованный Microsoft способ обеспечения автоматического запуска программного кода. Подробнее про работу с событиями будет рассказано в *гл. 4* и *5*;
- можно запустить приложение из командной строки с параметром `/m` и именем макроса, например:

  ```
  winword.exe /mMyMacros
  ```

  Очень удобно в этом случае использовать ярлыки, в которых можно указать этот параметр запуска.

В VBA вполне допустима ситуация, когда функция запускает на выполнение саму себя. Однако подобных вызовов лучше избегать (по возможности заменяя их на циклы). Причина — проблемы с читаемостью и возможное (в случае бесконечного запуска) исчерпание оперативной памяти (переполнение стека), чреватое серьезными системными ошибками.

Для завершения выполнения процедуры в VBA предусмотрены конструкции `End` и `Exit`. Синтаксис их очень прост:

```
Exit Sub
End Sub
```

Делают они одно и то же — завершают работу текущей процедуры. Однако используются в разных ситуациях:

- оператор `End` — это завершение работы процедуры после того, как все сделано. После оператора `End` код процедуры заканчивается;
- оператор `Exit` — это немедленное завершение работы функции в ходе ее работы. Обычно помещается в блок оператора условного перехода, чтобы произвести выход, как только выяснилось, что функции по каким-то причинам дальше выполняться не нужно (например, дальше идет код обработчика ошибок).

Задание для самостоятельной работы 3.4: Работа с процедурами и функциями

ЗАДАНИЕ:

1. Создайте в модуле **NewMacros** шаблона Normal.dot новую функцию `fMultiply()`, которая бы:

- принимала в качестве входящих параметров два числа;
- перемножала их и возвращала полученное значение.

2. Создайте в модуле **NewMacros** шаблона Normal.dot новую процедуру `AutoNew()` со следующим кодом:

```
Public Sub AutoNew()
    Dim nMult1 As Integer
    Dim nMult2 As Integer
    Dim nResult As Integer
    nMult1 = CInt(InputBox("Введите первое число: "))
    nMult2 = CInt(InputBox("Введите второе число: "))
    nResult = 10
    Selection.InsertAfter nResult
    Selection.Collapse wdCollapseEnd
End Sub
```

3. Измените процедуру `AutoNew()` таким образом, чтобы она передавала значения переменных `nMult1` и `nMult2` функции `fMultiply()` и принимала от нее значение для переменной `nResult` (это значение должно использоваться вместо жестко определенного значения `10`).
4. Создайте в Word новый документ и убедитесь, что созданные вами процедуры и функции работают правильно.
5. Чтобы созданная вами процедура `AutoNew()` не мешала дальнейшей работе, закомментируйте весь ее код.

Ответ к заданию 3.4

1. Запустите Word и нажмите клавиши <Alt>+<F11>. В окне **Project Explorer** раскройте узел **Normal | Modules** и щелкните два раза левой кнопкой мыши на строке **NewMacros**.
2. Вставьте в модуль **NewMacros** следующие строки для функции `fMultiply()`:

```
Public Function fMultiply(nM1 As Integer, nM2 As Integer)
    fMultiply = nM1 * nM2
End Function
```

3. Код для процедуры `AutoNew()` может выглядеть так (измененный код выделен полужирным):

```
Public Sub AutoNew()
    Dim nMult1 As Integer
    Dim nMult2 As Integer
```

```
    Dim nResult As Integer
    nMult1 = CInt(InputBox("Введите первое число: "))
    nMult2 = CInt(InputBox("Введите второе число: "))
    nResult = fMultiply(nMult1, nMult2)
    Selection.InsertAfter nResult
    Selection.Collapse wdCollapseEnd
End Sub
```

4. Для того чтобы закомментировать код `AutoNew()`, выделите весь код этой процедуры (включая `Public Sub AutoNew()` и `End Sub`) и нажмите кнопку **Comment Block** на панели инструментов **Edit**.

3.9. Встроенные функции языка VBA

3.9.1. Что такое встроенные функции

В языке программирования VBA предусмотрено несколько десятков *встроенных функций*. Они доступны в любой программе на языке VBA, при этом безразлично, в среде какого программного продукта мы находимся — Excel, Word, Access или, к примеру, AutoCAD. Используются они очень активно, и во многих ситуациях без них не обойтись. Профессиональные программисты применяют их совершенно автоматически, а обычным пользователям хочется посоветовать потратить несколько часов на знакомство с ними, потому что без знания этих функций эффективно работать в VBA не получится. Дополнительным аргументом в пользу их изучения является то, что практически идентичный набор функций есть в обычном Visual Basic и VBScript, а многие из этих функций с теми же названиями и синтаксисом встречаются и в других языках программирования — C++, Delphi, Java, JavaScript и т. п.

В справке по VBA встроенные функции сгруппированы по буквам (рис. 3.2).

Многие слушатели на курсах задавали вопрос: а нет ли справки по этим функциям на русском языке? К сожалению, такой справки мне найти не удалось, поэтому попытаюсь привести краткую справку в этой книге. Далее будет рассказано про большинство активно используемых функций языка VBA (математические функции, такие как косинус или тангенс, которые в практической работе почти не используются, и финансовые функции мы рассматривать не будем). Полный синтаксис функций для экономии места приводиться не будет: главное — понимание, что делает каждая функция и в каких ситуациях ее можно использовать.

Функции в следующих разделах сгруппированы по своей функциональности. Если нужно найти информацию просто по имени функции, можно воспользоваться предметным указателем в конце книги.

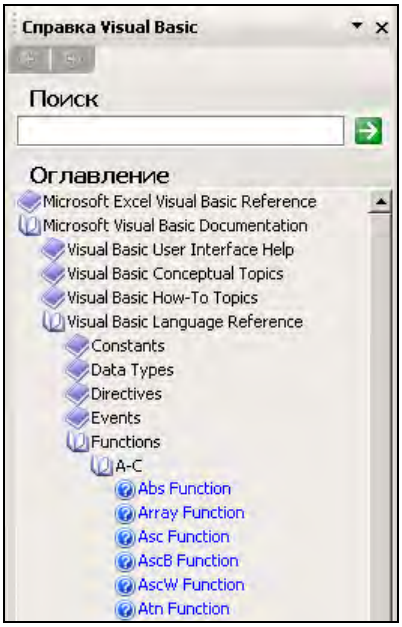


Рис. 3.2. Справка по встроенным функциям

3.9.2. Функции преобразования и проверки типов данных

В программах на VBA очень часто приходится преобразовывать значения из одного типа данных в другой. Приведу несколько типичных ситуаций, когда этим приходится заниматься:

- преобразование из строкового значения в числовое при приеме значения от пользователя через `InputBox()`;
- преобразование значения даты/времени в строковое, когда нам нужно отобразить дату или время единообразно вне зависимости от региональных настроек на компьютерах пользователей;
- преобразование значения из строкового в дату/время для применения специальных функций даты/времени.

Чаще всего для конвертации типов данных используются функции, имя которых складывается из префикса `'C'` (от слова *Convert*) и имени типа данных. Перечень этих функций следующий: `CBool()`, `CByte()`, `CCur()`, `CDate()`, `CDbl()`, `CDec()`, `CInt()`, `CLng()`, `CSng()`, `CStr()`, `CVar()`, `CVDate()`, `CVErr()`.

Просмотреть, что в итоге получилось, можно при помощи функции `TypeName()`, которая возвращает имя используемого типа данных, например:

```
nVar1 = CInt(InputBox("Введите значение"))
MsgBox TypeName(nVar1)
```

В данном случае эта функция вернет "Integer".

Кроме того, существует еще несколько полезных для конвертации функций.

- Str() — позволяет перевести числовое значение в строковое. Делает почти то же самое, что и CStr(), но при этом вставляет пробел перед положительными числами.
- Val() — "вытаскивает" из смеси цифр и букв только числовое значение. При этом функция читает данные слева направо и останавливается на первом нечисловом значении (допускается единственное нечисловое значение — точка, которая будет отделять целую часть от дробной). Очень удобно, когда у нас вперемежку с числовыми данными прописываются единицы измерения или валюта.
- IsNumeric() и IsDate() — проверяют значения на соответствие, чтобы не возникло ошибок при конвертации. Для проверки на соответствие специальным значениям можно использовать функции IsArray(), IsEmpty(), IsError(), IsMissing(), IsNull() и IsObject(). Все эти функции возвращают True или False в зависимости от результатов проверки переданного им значения.
- Hex() и Oct() — преобразовывают десятичные данные в строковое представление шестнадцатеричных и восьмеричных значений.

3.9.3. Строковые функции

Это наиболее часто используемые функции. Требуются они постоянно, и необходимо знать их очень хорошо.

- Asc() — эта функция позволяет вернуть числовой код для переданного символа. Например, Asc("D") вернет 68. Эту функцию удобно использовать для того, чтобы определить следующую или предыдущую букву. Обычно она используется вместе с функцией Chr(), которая производит обратную операцию — возвращает символ по переданному его числовому коду. Например, такой код в Excel позволяет написать в ячейки с A1 по A20 последовательно буквы русского алфавита от А до У:

```
Dim n, nCharCode As Integer
n = 1
nCharCode = Asc("A")
Do While n <= 20
    ActiveWorkbook.ActiveSheet.Range("A" & n).Value = Chr(nCharCode)
```

```
    n = n + 1
    nCharCode = nCharCode + 1
Loop
```

Варианты этой функции — `AscB()` и `AscW()`. `AscB()` возвращает только первый байт числового кода для символа, а `AscW()` возвращает код для символа в кодировке Unicode.

- `Chr()` — возвращает символ по его числовому коду. Помимо того, что используется в паре с функцией `Asc()` (см. предыдущий пример), без нее не обойтись еще в одной ситуации: когда нужно вывести служебный символ. Например, нам нужно напечатать в Word значение `"Газпром"` (в кавычках). Кавычка — это служебный символ, и попытка использовать строку вида:

  ```
  Selection.Text = ""Газпром""
  ```

  приведет к синтаксической ошибке. А вот так все будет в порядке:

  ```
  Selection.Text = Chr(34) & "Газпром" & Chr(34)
  ```

  Есть варианты этой функции — `ChrB()` и `ChrW()`. Работают аналогично таким же вариантам для функции `Asc()`.

- `InStr()` и `InStrRev()` — одни из самых популярных функций. Позволяют обнаружить в теле строковой переменной последовательность символов и вернуть ее позицию. Если последовательность не обнаружена, то возвращается 0. Функция `InStr()` ищет с начала строки, а `InStrRev()` — с конца.
- `Left()`, `Right()`, `Mid()` — позволяют взять указанное вами количество символов из существующей строковой переменной слева, справа или из середины соответственно.
- `Len()` — возвращает число символов в строке (длину строки). Часто используется с циклами, операциями замены и т. п.
- `LCase()` и `UCase()` — переводят строку в нижний и верхний регистры соответственно. Часто используются для подготовки значения к сравнению, когда регистр не важен (фамилии, названия фирм, городов и т. п.).
- `LSet()` и `RSet()` — заполняют одну переменную символами другой без изменения ее длины (соответственно слева и справа). Лишние символы обрезаются, на место недостающих подставляются пробелы.
- `LTrim()`, `RTrim()`, `Trim()` — убирают пробелы соответственно слева, справа или и слева, и справа.
- `Replace()` — заменяет в строке одну последовательность символов на другую.
- `Space()` и `String()` — возвращают строку из указанного вами количества пробелов или символов соответственно. Обычно используются для форма-

тирования вывода совместно с функцией `Len()`. Еще одна похожая функция — `Spc()`, которая используется для форматирования вывода на консоль. Она размножает пробелы с учетом ширины командной строки.

- `StrComp()` — сравнивает две строки.
- `StrConv()` — преобразует строку (в Unicode и обратно, в верхний и нижний регистры, первую букву слов заглавной и т. п.).
- `StrReverse()` — "переворачивает" строку, разместив ее символы в обратном порядке.
- `Tab()` — еще одна функция, которая используется для форматирования вывода на консоль. Размножает символы табуляции в том количестве, в котором вы укажете. Если никакое количество не указано, просто вставляет символ табуляции. Для вставки символа табуляции в строковое значение можно также использовать константу `vbTab`.

3.9.4. Функции для работы с числовыми значениями

Функций для работы с числовыми значениями в VBA очень много. Используются они реже, чем строковые функции, но во многих ситуациях без них не обойтись.

Еще один момент: если вы программируете на языке VBA, то, скорее всего, на вашем компьютере установлен Microsoft Office с Excel. В Excel есть свой собственный мощный набор встроенных функций для работы с числовыми значениями, которые вполне доступны из VBA. Если вы в моем списке не найдете ничего подходящего для вашей ситуации, возможно имеет смысл воспользоваться функциями Excel.

Кроме того, если в окне **Надстройки** (меню **Сервис | Надстройки**) установить флажок напротив строки **Пакет анализа**, в Excel будет добавлен дополнительный набор аналитических научных и финансовых функций, а если в том же окне установить флажок напротив **Analysis ToolPak — VBA**, то эти функции станут доступны из VBA (только внутри Excel, в котором установлена эта надстройка).

Далее приведены только универсальные функции VBA для работы с числовыми значениями. Эти функции доступны из любых приложений VBA.

- `Abs()` — эта функция возвращает абсолютное значение переданного ей числа (то же число, но без знака). Например, `Abs(3)` и `Abs(-3)` вернут одно и то же значение 3. Обычно используется тогда, когда нам нужно определить разницу между двумя числами, но при этом мы не знаем, какое число — первое или второе — больше. Результат вычитания может быть и

положительным и отрицательным. Чтобы он был только положительным, используется эта функция.

- `Int()`, `Fix()` и `Round()` — позволяют по-разному округлять числа. `Int()` возвращает ближайшее меньшее целое, `Fix()` отбрасывает дробную часть, `Round()` округляет до указанного количества знаков после запятой. При этом `Round()` работает не совсем правильно, в чем легко убедиться:

  ```
  MsgBox Round(2.505, 2)
  ```

  Поэтому на практике для округления лучше использовать `Format()` *(см. разд. 3.9.6)*:

  ```
  MsgBox Format(2.505, "#,##0.00")
  ```

- `Rnd()` и команда `Randomize` — используются для получения случайных значений (очень удобно, например, при генерации имен файлов). Обычный синтаксис при применении `Rnd()` выглядит так:

  ```
  случайное_число = Int(минимум + (Rnd() * максимум))
  MsgBox (Int(1 + (Rnd() * 100)))
  ```

  Однако перед вызовом функции `Rnd()` необходимо выполнить команду `Randomize` для инициализации генератора случайных чисел.

- `Sgn()` — позволяет вернуть информацию о знаке числа. Возвращает 1, если число положительное, –1, если отрицательное, и 0, если проверяемое число равно 0.

3.9.5. Функции для работы с датой и временем

Без функций даты и времени обычно обойтись очень сложно. Самые важные функции VBA для работы с датой/временем приведены далее.

- `Date()`, `Time()`, `Now()` — возвращают соответственно текущую системную дату, текущее системное время и дату и время одновременно. Установить их можно при помощи одноименного соответствующего оператора, например, так:

  ```
  Date = #5/12/2004#
  ```

- `DateAdd()` — добавляет к дате указанное количество лет, кварталов, месяцев и так далее до секунд.
- `DateDiff()` — возвращает разницу между датами (в единицах от лет до секунд).
- `DatePart()` — очень важная функция, которая возвращает указанную вами часть даты (например, только год, только месяц или только день недели).

- `DateSerial()` — создает значение даты на основе передаваемых символьных значений. То же самое делает функция `DateValue()`, но при другом формате принимаемых значений. Аналогичным образом (для времени) работают `TimeSerial()` и `TimeValue()`.
- `Day()` (а также `Year()`, `Month()`, `Weekday()`, `Hour()`, `Minute()`, `Second()`) — специализированные заменители функции `DatePart()`, которые возвращают нужную вам часть даты.
- `MonthName()` — возвращает имя месяца словами по его номеру. Возвращаемое значение зависит от региональных настроек. Если они русские, то вернется русское название месяца.
- `Timer()` — возвращает количество секунд, прошедших с полуночи.

Если нужно получить дополнительные возможности работы с датой/временем, то в вашем распоряжении объектная модель `Outlook`. Например, при помощи ее можно получить информацию о праздниках и рабочих/нерабочих днях большинства стран мира. Подробнее — в *гл. 13*.

3.9.6. Функции для форматирования данных

Для форматирования данных в вашем распоряжении функция `Format()` и целый набор функций, которые начинаются с префикса `Format...` (`FormatNumber()`, `FormatCurrency()`, `FormatDateTime()` и т. п.) Синтаксис функции `Format()` выглядит так:

```
Format(выражение, "формат")
```

Эта функция принимает *выражение* и форматирует его в соответствии с параметром *формат*.

Несколько примеров использования `Format()` (посмотрите сами, что получится):

```
Format(15/20, "Percent")
Format(Date, "Long Date")
Format(1, "On/Off")
Format(334.9, "###0.00")
Format("Просто текст", ">" )
```

Для остальных функций `Format...` то, что они делают, понятно из названий.

Особая ситуация — когда нужно, чтобы дата отображалась на компьютерах пользователей единообразно вне зависимости от региональных настроек. В качестве решения можно использовать функцию `DatePart()`, при помощи ее перевести дату "по частям" в текстовый формат и склеить нужным образом.

3.9.7. Функции для организации взаимодействия с пользователем

Во многих программах VBA необходимо обеспечить взаимодействие с пользователем — проинформировать его о чем-то и, возможно, получить от него ответную реакцию. В принципе, для пользователя можно просто вывести текст в окне приложения (например, в текущем документе Word) или воспользоваться формой и элементами управления. Как это делается — мы узнаем в *гл. 5*, посвященной работе с формами и элементами управления, и в *гл. 10—15*, в которых речь пойдет об объектных моделях приложений Office. В этом разделе мы рассмотрим только применение для этой цели встроенных функций VBA.

Самой простой способ вывести информацию пользователю — воспользоваться встроенной функцией VBA `MsgBox()`. Примеров применения этой функции в нашей книге уже было множество, а полный ее синтаксис выглядит так:

```
MsgBox(Текст [, кнопки] [, заголовок_окна] [, файл_справки,
метка_в_файле_справки])
```

Возможностей у `MsgBox()` достаточно много:

- можно отображать разное количество кнопок (**OK**, **Cancel**, **Abort**, **Retry**, **Ignore**, **Yes**, **No**);
- можно показывать символы *Critical* (красный круг с крестом), *Exclamation* (восклицательный знак в желтом треугольнике), *Question* (вопросительный знак), *Information* (буква "I");
- можно выбирать кнопку по умолчанию;
- можно делать окно модальным или обычным.

В зависимости от того, на какую кнопку нажал пользователь, функция возвращает соответствующее значение (всего 7 вариантов). Подробнее читайте в справке по VBA. Пример возврата значения от `MsgBox()` может быть таким:

```
Dim nVar As Integer
nVar = MsgBox ("Будем делать?", vbInformation + vbOKCancel, _
               "Демонстрационное окно сообщения")
```

Если значение `nVar` равно 1, то пользователь нажал **OK**, если 2, то **Отмена** (Cancel).

Иногда (например, при пакетной обработке данных) хотелось бы, чтобы окно сообщения через некоторое время закрывалось само собой. Это можно сделать при помощи метода `Popup()` объекта `Wscript.Shell`. Для этого в проект через меню **Tools | References** нужно добавить ссылку на Windows Script Host Object Model, а после этого использовать следующий код:

```
Dim oShell As New WshShell
oShell.Popup "Test", 5
```

В остальном функциональность получившего окна одинакова с `MsgBox()`. Код возврата, если пользователь не нажал ни на какую кнопку, равен –1.

Самый простой способ принять информацию от пользователя — воспользоваться функцией `InputBox()`. Все очень просто:

```
Dim InputName
InputName = InputBox("Введите Ваше имя")
MsgBox ("Вы ввели: " & InputName)
```

Для `InputBox()` можно указать текст приглашения, заголовок окна, значение по умолчанию, местонахождение окна и файл справки. Не забывайте, что все вводимое пользователем `InputBox()` автоматически переводит в тип данных `String`, может потребоваться преобразование.

Можно привлечь внимание пользователя звуковым сигналом. Для этой цели используется оператор `Beep`:

```
Dim i
For i = 1 To 3
    Beep
Next i
```

Если нужно обеспечить более сложное взаимодействие с пользователем, можно подумать о применении формы, возможностей самого документа Office или помощника (Office Assistant). Очень мощные возможности обеспечивает и применение объектной модели Internet Explorer.

3.9.8. Функции — заменители синтаксических конструкций

В VBA предусмотрено несколько функций, которые позволяют заменять синтаксические конструкции условного перехода, например `If...Then...Else` или `Select Case`. Каких-то преимуществ применение этих функций не дает (может быть, код станет на несколько строчек короче), но профессиональные программисты очень любят их использовать.

Начинающим программистам рекомендуется применять обычные синтаксические конструкции, чтобы не путаться. Однако для чтения чужого кода необходимо знать и эти функции.

- `Choose()` — принимает число (номер значения) и список значений. Возвращает значение из списка, порядковый номер которого соответствует передаваемому числу.

Например, вызов функции:

```
Choose(2, "Первый", "Второй", "Третий")
```

вернет "Второй".

- `IIf()` — расшифровывается как *Immediate If*, т. е. "немедленный *If*". Представляет собой упрощенный вариант `If...Else`, когда проверяется условие и возвращается одно из двух значений. Например:

```
IIf(n > 10, "Больше десяти", "Меньше или равно десяти")
```

- `Switch()` — принимает неограниченное количество пар типа "выражение = значение", проверяет каждое выражение на истинность и возвращает значение для первого выражения, которое оказалось истинным. Например:

```
Function Language (CityName As String)
    Language = Switch(CityName = "Москва", "русский", CityName = _
        "Париж", "французский", CityName = "Берлин", "немецкий")
End Function
```

3.9.9. Функции для работы с массивами

Как уже говорилось, при программной работе с приложениями Microsoft Office массивы используются редко. Вместо них применяются коллекции. Однако в VBA предусмотрены возможности и для работы с массивами.

- `Array()` — позволяет автоматически создать массив нужного размера и типа и сразу загрузить в него переданные значения:

```
Dim myArray As Variant
myArray = Array(10,20,30)
MsgBox A(2)
```

- `Filter()` — позволяет на основе одного массива получить другой, отфильтровав в исходном массиве нужные нам элементы.
- `LBound()`, `UBound()` — возвращают соответственно информацию о нижней границе массива (номер первого имеющегося в массиве значения) и о верхней границе (номер последнего имеющегося значения).
- `Join()` — соединяет множество строк, составляющих массив, в одну строковую переменную. В качестве разделителя по умолчанию используется пробел, но можно указать и свой разделитель. Обратная функция, создающая массив из одной строки, — `Split()`. Эти функции очень удобны, например, при обработке значений, полученных из базы данных, электронной таблицы, макетного файла и т. п.

3.9.10. Функции для работы с файловой системой

В VBA предусмотрен набор встроенных функций для выполнения различных операций с файлами, каталогами, дисками и прочими объектами файловой системы. Информация об этих функциях приведена далее. Но не забывайте, что помимо этих функций (общих для всех приложений, в которых используется VBA) у нас есть, во-первых, возможности, специфические для данного приложения (например, открытие и сохранение документа Word средствами объектной модели Word). Во-вторых, на любом компьютере под управлением Windows есть объектная библиотека Microsoft Scripting Runtime, очень простая и удобная для выполнения различных операций с файлами, каталогами и дисками. Можно добавить в проект VBA ссылку на нее и использовать все имеющиеся в ней возможности. Если, к примеру, мне нужно пройтись по всем файлам в данном каталоге и что-нибудь с ними сделать (например, загрузить в Excel все файлы отчетов, которые пришли из филиалов), я использую именно эту библиотеку. Справку по ней можно найти на сайте Microsoft (**www.microsoft.com/scripting**).

Далее приведены встроенные функции для работы с файловой системой, предусмотренные в VBA.

- `CurDir()` — функция, которая возвращает путь к текущему каталогу, в котором будут сохраняться файлы вашего приложения по умолчанию.
- `Dir()` — позволяет искать файл или каталог по указанному пути на диске.
- `EOF()` — при операции чтения или записи в файл на диске эта функция вернет `True`, если вы находитесь в конце файла.
- `Error()` — позволяет вернуть описание ошибки по ее номеру. Генерировать ошибку нужно при помощи метода `RaiseError()` специального объекта `Err` (*см. гл. 6*, в которой рассказывается про перехват ошибок и отладку).
- `FileAttr()` — позволяет определить, как именно был открыт вами файл в файловой системе: на чтение, запись, добавление, в двоичном или текстовом режиме и т. п.
- `FileDateTime()` — позволяет получить информацию о последнем времени обращения к указанному вами файлу. Если к файлу после создания ни разу не обращались, то функция вернет время создания файла.
- `FileLen()` — возвращает длину указанного вами файла в байтах.
- `FreeFile()` — позволяет определить следующую свободную цифру, которую можно использовать как номер файла при его открытии.
- `GetAttr()` — позволяет обратиться к файлу и получить информацию о его атрибутах (скрытый, доступен только для чтения, архивный и т. п.).

- `Input()` — позволяет считать информацию из открытого файла. Например, считать информацию из файла C:\text1.txt и вывести ее в окно сообщений можно так:

```
Dim MyChar
'Открываем файл функцией Open() на чтение
Open "c:\text1.txt" For Input As #1
Do While Not EOF(1)   ' Пока файл не кончился,
    ' получаем по одному символу и добавляем его к предыдущим
    MyChar = MyChar & Input(1, #1)
Loop
Close #1          'Закрываем файл
MsgBox MyChar     'Выводим его содержание в окно сообщения
```

  Вариант этой функции — `InputB()` — позволяет указать количество байт, которые надо считать из файла.

- `Loc()` — от *Location* (местонахождение) — возвращает число, которое определяет текущее место вставки или чтения в открытом файле. Похоже работает функция `Seek()`, но она возвращает информацию о позиции, с которой будет выполняться следующая операция чтения или вставки.

- `LOF()` — от *length of file* — позволяет определить длину открытого файла в байтах.

`Open` — это не функция, а команда VBA, но без нее операции чтения и записи с файлами на диске не произвести. Справку по ней можно найти по словосочетанию "Open Statement". Как минимум, ей нужно передать имя открываемого файла, режим открытия и номер файла (номер файла — это его идентификатор для передачи другим функциям, его назначаете вы сами). Например, чтобы открыть файл на чтение с возможностью одновременного обращения к нему других пользователей, можно использовать код вида:

```
Open "c:\file1.txt" For Output Shared As #1
```

3.9.11. Другие функции VBA

В этот раздел попали те встроенные функции языка VBA, которые я не смог отнести ни к одной другой категории.

- `DoEvents()` — это очень важная функция. Она позволяет на время отвлечься от выполнения какой-то операции VBA и передать управление операционной системе, чтобы обработать накопившиеся в операционной системе события (например, нажатия клавиш пользователем). После этого продолжение операции VBA продолжается. Если у вас выполняется очень длительная операция (поиск на дисках, обработка большого объема дан-

ных и т. п.) и вы хотите дать пользователю возможность быстро прервать эту операцию, можно выполнять эту команду, например, каждый раз после обработки определенной "порции" данных.

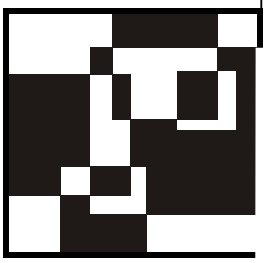
- `Environ()` — возвращает абсолютный путь для переменных окружения компьютера (полный список переменных, доступных на вашем компьютере, можно просмотреть, если в командной строке выполнить команду `SET`). Например, вам нужно записать что-то в файл во временном каталоге. Абсолютный путь к временному каталогу на вашем компьютере можно получить так:

```
MsgBox Environ("TEMP")
```

- `GetAllSettings()` — позволяет получить (в виде двумерного массива) из реестра все параметры, которые относятся к указанному вами приложению. Функция `SaveSetting()` позволяет записать информацию в реестр, а `DeleteSetting()` — удалить. `GetSetting()` позволяет получить информацию об определенном параметре. Замечу, что эти методы позволяют обращаться только к одному очень далекому уголку реестра в ветви `HKEY_CURRENT_USERS`. Обращаться к другим параметрам реестра при помощи этих методов бесполезно. Рекомендую для работы с реестром использовать объектную библиотеку Windows Script Host Object Model, которая также есть на любом компьютере под управлением Windows 2000, XP и 2003. Нужный объект называется `WSHShell`, методы — `RegRead()`, `RegWrite()` и `RegDelete()`. Справку по объектам этой библиотеки можно найти на сайте Microsoft (**www.microsoft.com/scripting**).
- `Partition()` — определяет, к какому диапазону из наборов значений относится переданное вами число, и возвращает описание этого диапазона (в виде строки). Обычно используется при выполнении запросов к базам данных.
- `QBColor()` — позволяет перевести обозначение цвета из старого номерного обозначения с возможными 16 значениями в RGB-код, который понимает VBA. Обычно используется при исправлении старых унаследованных программ.
- `RGB()` — еще одна функция для работы с цветом. Позволяет вернуть RGB-код, который можно использовать для присвоения цвета, приняв три значения для цветов: красного (Red), зеленого (Green) и синего (Blue). Значение для каждого из основных цветов могут варьироваться от 0 до 255. Например, самый зеленый из возможных цветов получится, если переданные этой функции значения будут выглядеть как `RGB(0, 255, 0)`.
- `Shell()` — позволяет запустить из VBA внешний программный файл и вернуть информацию о его Program ID в операционной системе. Обычно

применяется опытными разработчиками при использовании ими в программах возможностей Windows API. С практической точки зрения эту функцию можно использовать для запуска любых внешних программ из вашего приложения, хотя, с моей точки зрения, применение специальных объектов `WshShell` и `WshExec` из библиотеки Windows Script Host Object Model удобнее (можно передавать в окно клавиатурные комбинации, принимать и передавать значения через командную строку и т. п.). Эта библиотека есть на любом компьютере Windows, справку по ней можно найти на сайте **www.microsoft.com/scripting**.

- `TypeName()` — функция, которая возвращает имя типа данных для переданной ей переменной. Очень удобна для определения типа данных для значения, полученного из базы данных или путем вызова метода какого-то объекта.
- `VarType()` — делает почти то же самое, но вместо имени возвращает числовой код, который обозначает тип данных. Можно использовать для программных проверок типов данных для переменных.



ГЛАВА 4

Работа с объектами и объектные модели

В предыдущей главе вы познакомились с синтаксическими конструкциями и встроенными функциями языка VBA, но с одними встроенными функциями много не сделаешь. Вся функциональность программ VBA, работающих в приложениях Microsoft Office, полностью построена на работе с классами и объектами.

Объектно-ориентированное программирование — это отдельная очень большая тема, которой посвящено множество книг. В этой главе будут рассмотрены только те моменты, которые важны с точки зрения практической работы при программировании в среде Microsoft Office.

4.1. Что такое классы и объекты

Классы формально определяются как блоки функциональности, которые можно использовать в программах. Для наших целей их можно считать "чертежами" для создания объектов. На основе этих "чертежей" создаются экземпляры классов — объекты. Для простоты можно представить себе, что в оперативной памяти компьютера по чертежу построили дом — объект, с которым можно что-то делать.

Наборы чертежей (в терминологии программирования — коллекции классов) обычно называются библиотеками типов. В Windows они "упаковываются" в файлы DLL или OCX (иногда и в файлы других типов, например, EXE или TLB). Такие библиотеки типов откомпилированы — чертежи из них можно использовать, но просмотреть их (т. е. просмотреть исходный код класса) нельзя, для такой ситуации существует специальный термин — технология "черного ящика".

Чаще всего в программе VBA создается объект определенного класса (по-английски это называется *instantiation*, создание экземпляра — *instance*), и

далее работа производится с этим объектом. В одной программе вполне можно использовать несколько разных объектов одного и того же класса.

В этой книге рассказывается только про использование в программах объектов, созданных на основе уже готовых чертежей (которые приготовили для вас разработчики Microsoft Office). Создание своих собственных классов — это отдельная большая тема, которая выходит за рамки данной книги.

4.2. Создание и удаление объектов

Создание объекта в VBA может производиться разными способами. Самый простой способ выглядит так:

```
Dim oApp As Object
Set oApp = CreateObject("Word.Application")
MsgBox oApp.UserName
```

Это так называемое *позднее связывание* (late binding). Вначале мы объявляем переменную `oApp` с возможностью ссылаться на любой объект, а затем присваиваем ей ссылку на создаваемый нами объект `Word.Application`. Такое позднее связывание хуже с точки зрения производительности и расхода оперативной памяти, кроме того, редактор Visual Basic отказывается нам подсказывать, какие свойства и методы есть у этого объекта. Поэтому позднее связывание имеет смысл использовать только тогда, когда вы собираетесь хранить в этой переменной, согласно логике вашей программы, объекты разных типов. В остальных ситуациях предпочтительнее использовать *раннее связывание* (early binding):

```
Dim oApp As Word.Application
Set oApp = CreateObject("Word.Application")
MsgBox oApp.UserName
```

В этом случае мы сразу присваиваем переменной `oApp` тип `Word.Application`, а потом присваиваем ей ссылку на создаваемый нами объект. Можно обойтись и без второй строки:

```
Dim oApp As New Word.Application
MsgBox oApp.UserName
```

Однако ключевое слово `New` не может использоваться при создании зависимых объектов, с ключевым словом `WithEvents` и при создании переменных встроенных типов (`String`, `Integer` и т. п.), поэтому иногда необходимо использовать только объявление с `Set`. Кроме того, в языке VBScript синтаксической конструкции `New` нет, поэтому если вы собираетесь использовать оба языка, лучше сразу привыкать к конструкции `CreateObject()`. Но поскольку

конструкция с `New` короче, в VBA традиционно чаще используется именно она.

Еще одна возможность создания объекта — *воспользоваться методом другого объекта*, который создаст нужный нам объект и возвратит ссылку на него напрямую или через коллекцию:

```
Dim oApp As New Word.Application
oApp.Documents.Add
Dim oDoc As Word.Document
Set oDoc = oApp.Documents(1)
oDoc.SaveAs "C:\docvba1.doc"
```

В этом примере мы вначале создаем объект `Word.Application`, затем при помощи метода `Add()` коллекции `Documents` создаем в этой коллекции новый документ, потом получаем на него ссылку для переменной `oDoc`, а потом вызываем метод `SaveAs()` созданного нами объекта документа.

Удаление объектов производится очень просто:

```
Set Объектная_переменная = Nothing
```

например:

```
Set oDoc = Nothing
```

В принципе, объект можно и не удалять — он будет удален автоматически после того, как последняя объектная переменная, которая на него ссылается, уйдет за область видимости (обычно когда закончит работу процедура, в которой он используется). Однако явное удаление объектов — это "правило хорошего тона", которое позволит вам при создании серьезных приложений избежать конфликтов имен и перерасходования ресурсов.

Еще один момент, связанный с удалением объектов. Не все объекты можно удалить при помощи синтаксической конструкции `Nothing`. Некоторые объекты требуют, чтобы их удаляли из памяти специальным способом. Например, для удаления из памяти объекта приложения Word, который мы создавали в нашем примере, нужно обязательно вызвать его метод `Quit()`, иначе он выдаст сообщение об ошибке.

4.3. Методы объекта

Как правило, объект нам нужен для того, чтобы воспользоваться его методами, свойствами или событиями.

Метод — это именованный набор действий, которые может выполнять данный объект. Он может выполнять какие-либо операции, принимать и возвращать значения.

Существует три способа вызова метода.

- Самый простой способ выглядит так:

  ```
  Объект.Метод
  ```

  например:

  ```
  oDoc.Activate
  ```

  При этом не возвращаются и не принимаются никакие параметры.

- Второй способ:

  ```
  Объект.Метод параметр1 [, параметр2, ... , параметрN]
  ```

  Параметры передаются путем перечисления через запятую. Например:

  ```
  oDoc.SaveAs "C:\doc12.doc"
  ```

  В этом случае мы игнорируем то, что возвращает метод и поэтому круглые скобки не нужны.

- Третий способ:

  ```
  Переменная = Объект.Метод(параметр1 [, параметр2, ... , параметрN])
  ```

  например:

  ```
  Dim nCent
  nCent = oApp.CentimetersToPoints(10)
  MsgBox nCent
  ```

  В этом случае значение, которое возвращает метод, присваивается переменной. При этом использовать круглые скобки для передаваемых параметров обязательно. Даже если никакие параметры не передаются, круглые скобки все равно обязательны:

  ```
  Переменная = Объект.Метод()
  ```

4.4. Свойства объекта

Свойства объекта — это возможность получения доступа к информации, которая хранится в этом объекте. Через свойства можно получить эту информацию или изменить ее.

Извлечь информацию можно при помощи синтаксиса:

```
Переменная = Объект.Свойство
```

например:

```
sName = oApp.UserName
```

Изменить информацию в объекте при помощи свойства можно так:

```
Объект.Свойство = Значение
```

например:

```
oApp.ActivePrinter = "HP LaserJet 4"
```

Значение может быть:

- обычной константой (`10` или `"HP LaserJet 4"`);
- простым выражением `(10 + 5)`;
- свойством другого объекта:

```
Объект1.Свойство = Объект2.Свойство
```

- возвращаемым значением какого-либо метода:

```
Объект1.Свойство = Объект2.Метод()
```

Конечно, значения не всех свойств можно изменять. Некоторые свойства доступны только для чтения, другие — для чтения и записи, третьи (очень редко) — только для записи.

4.5. События объекта и объявление *WithEvents*

Событие — это действие, распознаваемое объектом, для которого можно запрограммировать отклик. Например, в качестве события можно использовать открытие или закрытие документа, щелчок мыши, нажатие клавиши. События запрятаны в глубь объектов и настоятельно рекомендуется их использовать уже рассмотренным нами способом — через выбор нужного объекта и его события в окне редактора кода Visual Basic. Однако в некоторых ситуациях события для объектов не появляются в окне редактора кода (например, это справедливо для очень важного объекта `Application`). В этом случае необходимо явно объявить этот объект с событиями — при помощи ключевого слова `WithEvents`, например, так:

```
Public WithEvents App As Word.Application
```

Делается это в области объявлений модуля. После этого в редакторе кода Visual Basic появляется новый объект `App` со всеми необходимыми событиями.

Подробно работу с событиями мы рассмотрим в следующей *гл. 5*, которая будет посвящена работе с формами и графическими элементами управления: кнопками, флажками, переключателями и т. п. Нам достаточно выбрать в списке объектов (левый верхний список в окне редактора кода) нужный гра-

фический элемент, затем в списке событий (справа от списка объектов) выбрать нужное событие, и в редакторе кода будет автоматически создана специальная событийная процедура. Код, который вы в нее впишете, будет автоматически выполнен при наступлении этого события (например, при щелчке мышью на кнопке формы).

4.6. Просмотр объектов

Элементарные знания о том, как создавать объекты и использовать их свойства, методы и события, у вас уже есть. Однако может возникнуть вопрос: как найти нужный объект и как определить, какие свойства, методы и события в нем имеются?

Основной инструмент для этой цели — **Object Browser**, утилита, которая интегрирована в редактор кода VBA. Чтобы ей воспользоваться, необходимо в окне редактора кода нажать клавишу <F2> и выбрать нужную библиотеку классов. Классы показываются в левом списке как прямоугольники с разноцветными "кирпичиками", методы — как летящий зеленый предмет, свойства — как надпись, на которую указывает рука, события — значок молнии. Свои значки предусмотрены для модулей и перечислений. Если нужно просмотреть библиотеку типов, которой еще нет в списке **Object Browser**, необходимо добавить ссылку на нее через меню **Tools | References** или пункт **References** в контекстном меню самого **Object Browser**. Однако необходимо учитывать:

- для полноценной работы с **Object Browser** необходимо разбираться в объектно-ориентированном программировании. Например, если мы просмотрим класс `CommandButton` из библиотеки `MSForms` (т. е. класс кнопки на форме), то увидим там далеко не все его свойства, методы и события. Причина в том, что многие свойства, методы и события этот класс наследует от класса `Control` — общего прародителя большинства элементов управления VBA;
- при помощи **Object Browser** вы сможете узнать только названия методов, свойств и событий и получить информацию о принимаемых параметрах и возвращаемых значениях. Получить информацию о том, что делает данный метод, что возвращает свойство, когда срабатывает событие, нельзя (иногда можно догадаться по названию). Эту информацию можно найти только в справке по данной библиотеке классов.

4.7. Объектные модели

Наборы объектов, которые предназначены для выполнения задач, относящихся к одной области, называются объектными моделями. Например, в объ-

ектной модели Excel предусмотрены объекты, представляющие само приложение Excel, рабочую книгу, отдельные листы на этой рабочей книге, наборы ячеек, диаграммы и т. п. В *гл. 10—15* будут подробно разобраны объектные модели приложений Microsoft Office: Word, Excel, Access, PowerPoint, Project, Outlook. Однако при программировании на языке VBA и при создании своих собственных приложений ограничиваться только объектными моделями приложений Office совсем не обязательно.

В операционную систему Windows встроено множество других объектных моделей, применение которых может очень сильно расширить возможности ваших приложений. Далее приведен список дополнительных объектных моделей, которые встроены в Windows или в другие продукты Microsoft (об этом будет сказано отдельно), которыми я пользуюсь очень активно. Справку по большинству этих объектных моделей можно найти в MSDN (Microsoft Developer Network, официальная документация для разработчиков). Получить из нее информацию можно со страницы **www.microsoft.com/msdn**. Кроме того, MSDN можно установить на свой компьютер с компакт-дисков или DVD.

Чтобы использовать возможности этих объектных моделей в своей программе, необходимо добавить ссылку на них в ваш проект. Делается это очень просто: в окне редактора Visual Basic выберите **Tools | References** и в списке установите флажок около нужной библиотеки.

Вот перечень наиболее интересных с точки зрения применения в своих приложениях объектных моделей:

- *Windows Script Host Object Model* (wshom.exe) — эта библиотека предназначена для автоматизации работы администраторов. Она предоставляет возможность программно работать с сетью, принтерами, реестром, ярлыками, журналом событий, позволяет запускать внешние приложения и передавать в них нажатия клавиш и консольные строки и т. п. Есть на всех компьютерах с Windows 2000, XP, 2003 (в качестве необязательного компонента имеется и в Windows 98 Second Edition и Windows NT 4.0);
- *Microsoft Scripting Runtime* (scrrun.dll) — еще одна библиотека для администраторов. Главное ее богатство — очень удобный (и при этом простой) набор классов для работы с файловой системой — дисками, каталогами, файлами, содержимым текстовых файлов и т. п. Поставляется в одном наборе с Windows Script Host Object Model;
- *Microsoft ADO* (набор файлов, начинающихся с "msado") — классы для работы с базами данных. Эта библиотека будет подробно рассмотрена в *гл. 9*. Также имеется на всех без исключения компьютерах под управлением Windows 2000, XP и Windows 2003 Server (обычно сразу несколько версий);

- *Microsoft SQLDMO Object Library* (файл sqldmo.dll) — набор классов для получения полного контроля над Microsoft SQL Server (возможность производить любые административные операции, выполнять запросы и т. п.). Имеется только на тех компьютерах, на которых установлен SQL Server версий 7.0, 2000 или 2005. В SQL Server 2005 она разбита на несколько частей — SMO (SQL Server Management Objects), RMO (Replication Management Objects) и AMO (Analysis Management Objects);
- *Microsoft CDO* (версия 1.21, for NTS версия 1.2, for Windows 2000 версия 1.0; файлы olemsg.dll, cdonts.dll, cdosys.dll) — наборы классов для работы с электронной почтой. Можно использовать для создания и отправки своих почтовых сообщений, просмотра новых сообщений в почтовом ящике и т. п. Есть на всех компьютерах под управлением Windows 2000, XP и Windows 2003 Server. На компьютерах с более старыми операционными системами обычно также имеется, поскольку эта библиотека устанавливается вместе с Microsoft Office;
- *Microsoft WMI Scripting v1.1* (wbemdisp.tlb) — расширение возможностей программ через программный интерфейс WMI (Windows Management Instrumentarium). Возможности совершенно невероятные: от управления скоростью вращения вентилятора (и вообще работы с любыми устройствами, про которые знает операционная система) до установки программного обеспечения, запуска процессов на удаленном компьютере, управления службами и т. п. В этой модели реализованы очень мощные возможности работы с событиями. Например, можно выполнять какой-либо код в ответ на запуск или завершение работы программы с определенным именем на удаленном компьютере, в ответ на создание или удаление файла в каталоге, появления записи в журнале событий и т. п. Эта объектная модель (вместе со службой WMI) встроена во все компьютеры под управлением Windows 2000, XP и Windows 2003 Server. Тем, кого интересуют приемы работы с WMI, я могу посоветовать обратиться к своей статье (точнее, к главе из учебного курса), которая доступна по адресу:

  www.askit.ru/progr_admin/progr_admin_m14.htm;
- *Active Directory Scripting Interface* (adsldp.dll, wldap32.dll, adsnt.dll, adsnds.dll, adsnw.dll) — взаимодействие с объектами в каталогах Active Directory, NT, NetWare, т. е. работа с учетными записями пользователей, группами, объектами компьютеров, принтеров и т. п. Также встроена в операционные системы Windows 2000, XP и Windows 2003 Server (как в серверные, так и в пользовательские версии);
- *объектная модель Windows Explorer*. Информации о ней в справке не так много, но иногда она очень удобна для выполнения различных операций с файлами на диске;

- *объектная модель Internet Explorer* — очень мощное и удобное средство для организации взаимодействия с пользователем. Позволяет показывать пользователю Web-страницы (последовательно меняя их, можно организовать "мультфильм"), флэш-ролики, демонстрировать видео- и аудиоклипы и т. п. Эта объектная модель очень удобна и для сбора информации от пользователей при помощи скриптов и форм HTML, хотя в Microsoft Office для этой цели чаще всего используются обычные графические формы (о которых рассказывается в *гл. 5*).

Для тех, кто никогда не работал с этими объектными моделями, их применение может показаться темным лесом, но на практике все не так страшно. Они изначально создавались таким образом, чтобы с ними удобно было работать пользователям, не являющимся профессиональными программистами (например, администраторам). По опыту слушателей учебного курса "Программирование для администраторов", где рассматриваются эти объектные модели, освоить их можно очень быстро.

Задание для самостоятельной работы 4: Применение внешней объектной модели Windows Script Host в приложениях VBA

Подготовка:

Откройте Word и создайте новый документ (обычным, не программным способом). Затем нажмите клавиши <Alt>+<F11>, чтобы открыть редактор Visual Basic, в окне **Project Explorer** щелкните правой кнопкой мыши по контейнеру вашего документа (он должен называться **Project (Документ1)**) и в контекстном меню выберите **Insert | Module**. Будет создан новый стандартный модуль. При помощи меню **Insert | Procedure** создайте в нем новую процедуру с именем `WSH()`.

ЗАДАНИЕ:

1. Добавьте при помощи меню **Tools | References** в проект этого документа ссылку на библиотеку Windows Script Host Object Model.
2. В процедуре `WSH()` создайте программные объекты `WScript.Network` и `WScript.Shell` и просмотрите свойства и методы этих объектов.
3. Добавьте в процедуру `WSH()` код, который бы:
   - принимал в текстовые переменные и печатал в документе Word значения свойств `ComputerName`, `UserName` и `UserDomain` объекта `WScript.Network`;

Примечание

Код для вывода текстовой переменной в Word может выглядеть так:

```
ThisDocument.Activate
Selection.TypeText Text:=(текстовая_переменная)
```

- вызывал метод `Run()` объекта `WScript.Shell` и передавал ему единственный текстовый параметр со значением `"calc"`;
- использовал свойство `Environment` объекта `WScript.Shell` для создания коллекции текстовых переменных с информацией о переменных окружения;
- печатал в документе Word все значения текстовых переменных из этой коллекции.

Примечание

Для переменных, которые вы будете использовать для создаваемой коллекции и ее элементов, следует использовать тип `Variant`.

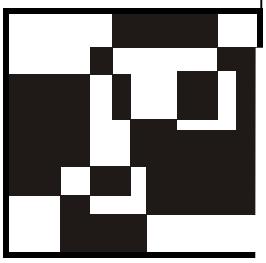
Ответ к заданию 4

Итоговый код процедуры `WSH()` может быть таким:

```
Public Sub WSH()
    Dim oNetwork As WshNetwork
    Dim oShell As WshShell
    Dim sComputer As String
    Dim sDomain As String
    Dim sUser As String
    Dim oColl As Variant
    Dim sEnv As Variant

    'Создаем объекты
    Set oNetwork = CreateObject("WScript.Network")
    Set oShell = CreateObject("Wscript.Shell")

    'Получаем и печатаем значения свойств объекта Wscript.Network
    sComputer = oNetwork.ComputerName
    sDomain = oNetwork.UserDomain
    sUser = oNetwork.UserName
    ThisDocument.Activate
    Selection.TypeText Text:=(sComputer & vbCrLf & sDomain & vbCrLf & _
                              sUser & vbCrLf & vbCrLf)
```

```
    'Вызываем метод Run объекта Wscript.Shell
    oShell.Run "Calc"

    'Получаем коллекцию переменных окружения
    Set oColl = oShell.Environment

    'И выводим каждый элемент этой коллекции
    For Each sEnv In oColl
        Selection.TypeText Text:=(sEnv & vbCrLf)
    Next

    'Правило хорошего тона — удаляем созданные объекты из памяти
    Set oNetwork = Nothing
    Set oShell = Nothing

End Sub
```

ГЛАВА 5



Формы, элементы управления и события

5.1. Для чего нужны формы

С самыми простыми возможностями организации взаимодействия с пользователем (применение встроенных функций `MsgBox()` и `InputBox()`) мы уже познакомились. Однако, конечно же, возможностей этих функций не всегда хватает. В этой главе речь пойдет о том, как создать графический интерфейс своего приложения с помощью VBA.

Чаще всего для предоставления пользователю графического интерфейса используются формы VBA. В принципе, многие элементы управления можно вставлять непосредственно на страницу документа (для этого используются панели инструментов **Формы** и **Элементы управления**), однако классический способ — это применение формы. Вне зависимости от того, используется ли форма или элементы управления размещаются напрямую в документе, набор элементов управления и приемы работы с ними одинаковы.

Как выглядит применение форм в приложении VBA? Обычно форма запускается при открытии пользователем документа. Пользователь выполняет на форме какие-то действия по вводу или выбору информации (например, выбирает значения в раскрывающемся списке, устанавливает значения для флажков и переключателей и т. п.), а потом, как правило, нажимает кнопку на этой форме, и введенная им информация передается в базу данных, отправляется по электронной почте, записывается в файл для распечатки и т. д.

5.2. Создание форм и самые важные свойства и методы форм

Создать форму очень просто: для этого достаточно в редакторе Visual Basic щелкнуть правой кнопкой мыши на проекте (т. е. на имени документа) в окне

Project Explorer и в контекстном меню выбрать **Insert | UserForm**. Откроется окно дизайнера форм (**Form designer**), в котором будет представлено пустое серое окно формы (по умолчанию она называется **UserForm1**) и рядом **Toolbox** — панель с набором элементов управления (рис. 5.1).

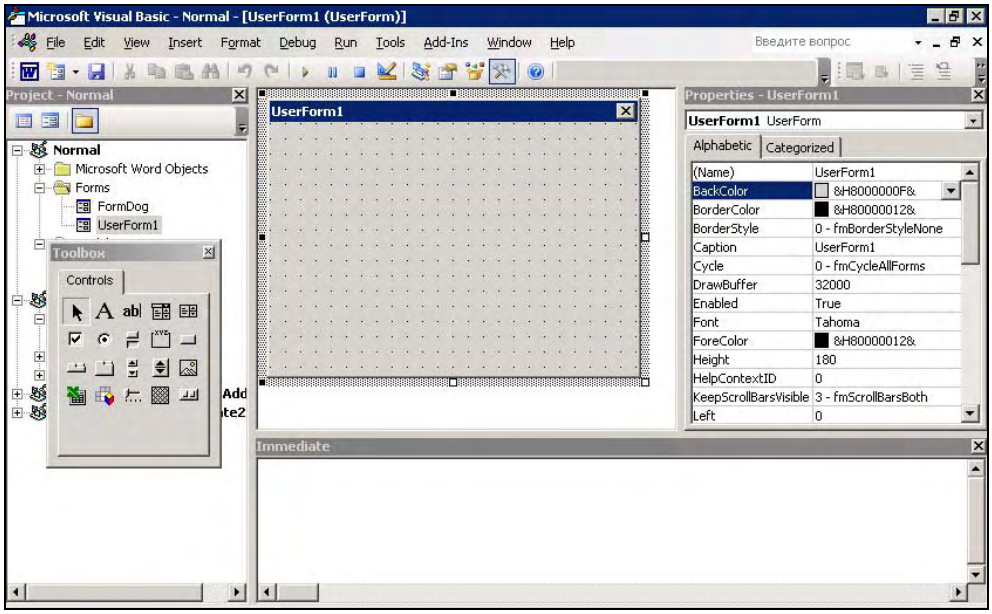


Рис. 5.1. Все готово для работы с формой

Если у вас включен показ окна свойств **Properties** (он включается по клавише <F4>), то в этом окне будут представлены свойства формы. Переход к редактору кода для этой формы (по умолчанию открывается событие `Click`) выполняется по клавише <F7>, возврат обратно в окно дизайнера форм — по <Shift>+<F7>.

Очень удобно, что для форм и элементов управления можно настраивать свойства при помощи графического интерфейса окна свойств — резко уменьшается количество программного кода, которое нужно писать вручную.

Некоторые самые важные свойства форм (кроме `ShowModal` все они применимы и для других элементов управления) приведены далее.

- `Name` — это свойство определяет имя формы. Пользователь вашей программы, скорее всего, его никогда не увидит. Имя формы используется только программистом в коде для этой формы (и в окнах редактора Visual Basic). После создания формы ее имя, предлагаемое по умолчанию (`UserForm1`), рекомендуется заменить на что-нибудь более значимое, чтобы

было проще ориентироваться в программе (это относится ко всем элементам управления).

- `Caption` — определяет заголовок формы (по умолчанию совпадает с именем формы). Рекомендуется ввести строку, которая будет напоминать пользователю о назначении формы (например, `"Выбор типа отчета"`).
- `Enabled` — если это свойство установлено в `False`, пользователь не сможет работать с формой. Используется для временного отключения формы, например, пока пользователь не обеспечит какие-то условия для ее работы.
- `ShowModal` — если свойство установлено в `True` (по умолчанию), то пользователь не может перейти к другим формам или вернуться в документ, пока не закроет эту форму (так называемый "модальный" режим работы).

Большая часть других свойств относится к внешнему виду, размерам и местонахождению формы.

Самые важные методы форм перечислены в следующем списке.

- В процессе редактирования формы (из окна редактора Visual Basic) ее можно запускать по нажатию клавиши <F5>. После того, как форма будет готова, вы должны обеспечить ее запуск в документе. Для запуска формы нужно воспользоваться методом `Show()`:

  ```
  UserForm1.Show
  ```

  Если форма уже была загружена в память, она просто станет видимой, если нет — то будет автоматически загружена (произойдет событие `Load`).

  Сам этот метод можно вызвать, например:

  - из обычного макроса, привязанного к кнопке или клавиатурной комбинации;
  - из автозапускаемого макроса (макроса с названием `AutoExec` для Word);
  - из кода для элемента управления, расположенного в самом документе (например, `CommandButton`) или на другой форме (для перехода между формами);
  - поместить его в обработчик события `Open` для документа Word или книги Excel, чтобы форма открывалась автоматически при открытии документа.
- После того, как пользователь введет или выберет нужные данные на форме и нажмет требуемую кнопку, форму необходимо убрать. Для этого можно воспользоваться двумя способами:

  - спрятать форму (использовать метод `Hide()`), например:

    ```
    UserForm1.Hide
    ```

Форма будет убрана с экрана, но останется в памяти. Потом при помощи метода `Show()` можно будет опять ее вызвать в том же состоянии, в каком она была на момент "прятанья", а можно, например, пока она спрятана, программно изменять ее и расположенные на ней элементы управления. Окончательно форма удалится из памяти при закрытии документа;

- если форма больше точно не потребуется, можно ее удалить из памяти при помощи команды `Unload`:

```
Unload UserForm1
```

Остальные методы относятся либо к обмену данными через буфер обмена (`Copy()`, `Cut()`, `Paste()`), либо к служебным возможностям формы (`PrintForm()`, `Repaint()`, `Scroll()`).

Важнейшая концепция VBA — события. Событие *(event)* — это то, что происходит с программой и может быть ею распознано. Например, к событиям относятся щелчки мышью, нажатия на клавиши, открытие и закрытие форм, перемещение формы по экрану и т. п. VBA построен таким образом, чтобы можно было создавать на нем программы, управляемые событиями *(event-driven)*. Такие программы противопоставляются устаревшему процедурному программированию.

Самые важные события форм приведены далее.

- `Initialize` — происходит при подготовке формы к открытию (появлению перед пользователем). Обычно в обработчик для этого события помещается код, связанный с открытием соединений с базой данных, настройкой элементов управления на форме, присвоением значений по умолчанию и т. п.
- `Click` (выбирается по умолчанию) и `DblClick` — реакция на одиночный и двойной щелчок мыши соответственно. Для формы эти события используются не так часто. Обычно обработчики щелчков применяются для кнопок (элементов управления `CommandButton`).
- `Error` — это событие используется при возникновении ошибки в форме, предоставляя пользователю возможность исправить сделанную им ошибку. Подробнее — в *гл. 6*, которая посвящена ошибкам и отладке.
- `Terminate` — используется при нормальном завершении работы формы и выгрузке ее из памяти (например, по команде `Unload`). Обычно применяется для разрыва открытых соединений с базой данных, освобождения ресурсов, протоколирования и т. п. Если работа формы завершается аварийно (например, запустившее форму приложение выдало команду `End`), то это событие не возникает.

Остальные события связаны либо с изменением размера окна формы, либо с нажатиями клавиш, либо с активизацией (получением фокуса) или деактивизацией (потерей фокуса).

Поскольку форма — это во многом просто контейнер для хранения других элементов управления, главное ее событие — `Initialize`. Все остальные события обычно используются не для формы, а для расположенных на ней элементов управления.

Нужно отметить некоторые моменты, связанные с созданием и редактированием форм:

- формы, создаваемые в Microsoft Access, не являются стандартными, как формы остальных приложений Office, и набор свойств и методов у них несколько отличается. Тем не менее по функциональности они практически одинаковы;
- иногда для обсуждения форму удобно распечатать. Для этого предусмотрено специальное диалоговое окно, которое можно вызвать по нажатию клавиш <Ctrl>+<P> (при выбранной форме в дизайнере);
- если все нужные вам элементы управления трудно уместить на одной форме (даже большого размера), в вашем распоряжении два варианта: воспользоваться двумя формами (осуществляя переход между ними при помощи методов `Show()` и `Hide()`, подвязанных к элементам управления) или воспользоваться несколькими вкладками для формы. Для этой цели в вашем распоряжении — специальный элемент управления `Multipage`.

5.3. Элементы управления

5.3.1. Что такое элемент управления

Элемент управления — это специализированный объект, который можно размещать на формах VBA (или непосредственно в документах) и который используется для организации взаимодействия с пользователем. В VBA есть как стандартные элементы управления (`CommandButton`, `CheckBox`, `OptionButton`), так и нестандартные (любые другие, которые есть на вашем компьютере, например, Microsoft Web Browser, представляющий Internet Explorer, элемент управления `Calendar` и т. п.). Элементы управления реагируют на события, которые генерирует пользователь (нажатие на кнопку, ввод значения, перемещение ползунка и т. п.).

Добавление элементов управления на форму чаще всего производится из дизайнера форм при помощи панели **Toolbox**. Для этого необходимо выбрать элемент управления на **Toolbox** и перетащить его на форму или, что более

удобно, выделить элемент управления в **Toolbox**, а затем на форме выделить ту область экрана, которую будет занимать этот элемент управления.

Добавление элементов управления можно производить и программным способом (при помощи метода `Add()` коллекции `Controls`), однако при этом вам придется указывать огромное количество свойств создаваемого элемента управления, что не очень удобно.

5.3.2. Элемент управления *Label*

Это самый простой элемент управления. *Надпись* (`Label`) — это просто область формы, в которой выводится какой-то текст (рис. 5.2).

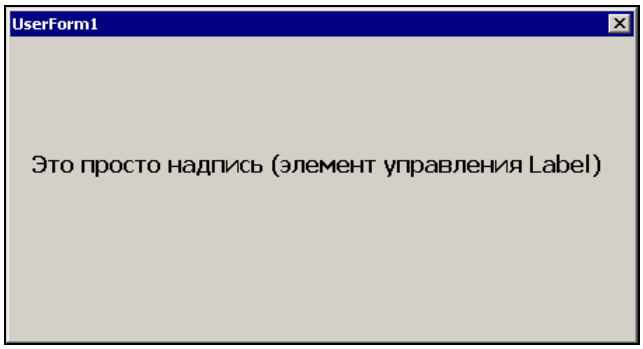


Рис. 5.2. Элемент управления `Label` на форме

Пользователь не может изменять этот текст. Чаще всего элемент управления `Label` используется как строка состояния с объяснением того, что сейчас произошло, или происходит, или должен сделать пользователь, и т. п. Этот элемент управления может использоваться и как пояснение для других элементов управления, таких как ползунок.

Главное свойство элемента управления `Label` — это `Caption`, тот текст, который будет выводиться на форме. Большая часть остальных свойств относится к форматированию этого текста или настройке внешнего вида этого элемента управления.

Несмотря на то, что для этого элемента управления предусмотрен набор событий (`Click`, `Error` и т. п.), использовать их не принято: пользователю обычно не приходит в голову, что по надписи нужно щелкнуть мышью.

5.3.3. Элемент управления *TextBox*

Текстовое поле (`TextBox`) — один из самых часто используемых элементов управления (рис. 5.3).

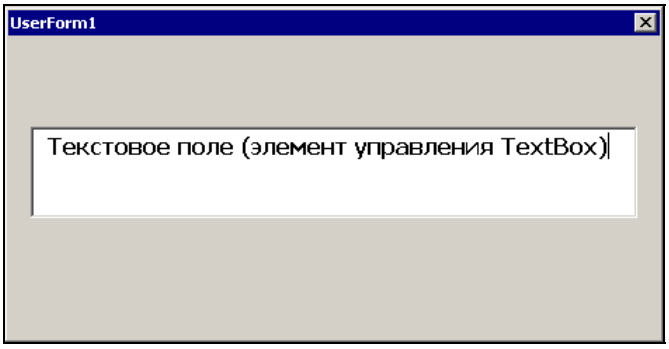


Рис. 5.3. Текстовое поле (элемент управления `TextBox`) на форме

Текстовое поле используется:

- для приема каких-либо текстовых данных, вводимых пользователем (например, для отправки по почте, для занесения в базу данных и т. п.);
- для вывода пользователю текстовых данных с возможностью их редактирования (из базы данных, листа Excel и т. п.);
- для вывода пользователю текстовых данных с возможностью копирования и печати, но без возможности изменения (классический пример — текст лицензионного соглашения).

Далее приведены некоторые важные свойства этого элемента управления.

- `Value` (или `Text`, эти два свойства для текстового поля идентичны) — то текстовое значение, которое содержится в этом поле. Используется для занесения исходного значения и для приема значения, введенного пользователем, в строковую переменную.
- `AutoSize` — позволяет текстовому полю автоматически менять свой размер, чтобы поместить весь текст. Использовать не рекомендуется, т. к. может нарушиться весь дизайн вашей формы.
- `ControlSource` — ссылка на источник текстовых данных для поля. Может ссылаться, например, на ячейку в Excel, на поле в объекте `Recordset` и т. п. При изменении пользователем данных в текстовом поле автоматически изменится значение на источнике, определенном в `ControlSource`.
- `ControlTipText` — текст всплывающей подсказки, которая появляется, когда пользователь наводит указатель мыши на элемент управления. Рекомендуется к заполнению для всех элементов управления (для самой формы это свойство не предусмотрено).
- `Enabled` — если установить в `False`, то текст в поле станет серым и с содержимым поля ничего нельзя будет сделать (ни ввести текст, ни выде-

лить, ни удалить). Обычно это свойство используется, чтобы показать пользователю, что этот элемент управления отключен до выполнения каких-либо условий (это относится ко всем элементам управления).

- `Locked` — поле будет выглядеть как обычно, пользователь сможет выделять и копировать данные из него, но не изменять их. Используется для показа неизменяемых данных типа лицензионных соглашений, сгенерированных значений и т. п.
- `MaxLength` — максимальная длина значения, которое можно ввести в поле. Иногда можно использовать свойство `AutoTab` — при достижении определенного количества символов управление автоматически передается другому элементу управления.
- `MultiLine` — определяет, можно ли использовать в текстовом поле несколько строк или только одну. Если вам нужно текстовое поле для приема одного короткого значения, подумайте, нельзя ли вместо элемента управления обойтись функцией `InputBox()`.
- `PasswordChar` — позволяет указать, за каким символом будут "прятаться" вводимые пользователем значения. Используется, конечно, при вводе пароля.
- `ScrollBars` — определяет, будут ли показаны горизонтальная и вертикальная полосы прокрутки (в любом сочетании). Если текст будет длинным, без них не обойтись.
- `WordWrap` — настоятельно рекомендуется включать в тех ситуациях, когда значение `MultiLine` установлено в `True`. В этом случае будет производиться автоматический переход на новую строку при достижении границы текстового поля.

Остальные свойства по большей части относятся к оформлению текстового поля и его содержания, а также к настройкам редактирования.

Главное событие для текстового поля — это событие `Change` (т. е. изменение содержания поля). Обычно на это событие привязывается проверка вводимых пользователем значений или синхронизация введенного значения с другими элементами управления (например, сделать доступной кнопку, изменить текст надписи и т. п.)

5.3.4. Элемент управления *ComboBox*

Комбинированный список (`ComboBox`) также используется очень часто. Этот элемент управления позволяет пользователю выбирать "готовые" значения из списка, так и вводить значения самостоятельно (хотя это можно запретить). Пример элемента управления `ComboBox` представлен на рис. 5.4.

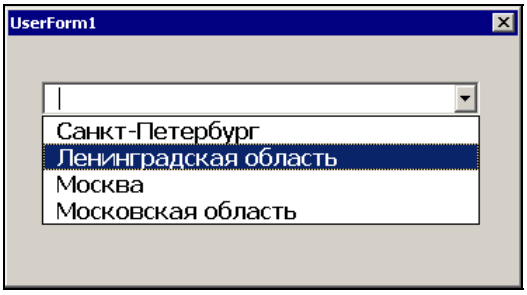


Рис. 5.4. Комбинированный список (элемент управления `ComboBox`) на форме

Обычно `ComboBox` используется в двух ситуациях:

- когда пользователю необходимо выбрать одно или несколько значений из списка размером от 4-х до нескольких десятков позиций. Если позиций меньше, то проще использовать переключатель (`OptionButton`), если больше — то ориентироваться в списке становится неудобно и необходимо использовать специальные приемы, когда пользователь вводит первые буквы нужного слова и в списке остаются только значения, которые начинаются с этих букв;
- когда список позиций для выбора необходимо формировать динамически на основании данных из источника (базы данных, листа Excel и т. п.).

К сожалению, через окно свойств заполнить список позициями не получится — для этой цели придется использовать специальный метод `AddItem()`. Обычно он помещается в обработчик события `Initialize` для формы. Применение его может выглядеть так:

```
Private Sub UserForm_Initialize()
    ComboBox1.AddItem "Санкт-Петербург"
    ComboBox1.AddItem "Ленинградская область"
    ComboBox1.AddItem "Москва"
    ComboBox1.AddItem "Московская область"
End Sub
```

Второй параметр `varIndex` (необязательный) этого метода может использоваться для определения положения элемента в списке, но он не может превышать значения свойства `ListCount` и поэтому для начальной загрузки `ComboBox` не подходит.

Самые важные свойства комбинированного списка представлены далее.

- `ColumnCount`, `ColumnWidth`, `BoundColumn`, `ColumnHeads`, `RowSource` — свойства, которые применяются при работе со списками из нескольких столбцов. Пользователи не любят такие списки, и поэтому к использованию они

не рекомендуются (гораздо проще сделать несколько комбинированных списков).

- `MatchEntry` — определяет, будут ли при вводе пользователем первых символов значения выбираться подходящие позиции из списка. Возможность очень удобная, рекомендуется сохранить значение, которое используется по умолчанию.
- `MatchRequired` — определяет, разрешается ли пользователю вводить то значение, которого нет в списке. По умолчанию `False`, т. е. разрешено.
- `Value` (или `Text`) — позволяет программным способом установить выбранное значение в списке или получить в переменную выбранное или введенное пользователем значение.

Остальные свойства (`AutoSize`, `Enabled`, `Locked`, `ControlText`, `ControlTipText`, `MaxLength`) применяются точно так же, как и для `TextBox`.

Главное событие для комбинированного списка — `Change`, то же, что и для `TextBox`. Обычно в обработчике этого события проверяются введенные пользователем значения, эти значения переносятся в текстовое поле или в `ListBox` (если нужно дать пользователю возможность выбрать несколько значений, поскольку свойства `MultiSelect` у `ComboBox` нет) и т. п.

5.3.5. Элемент управления *ListBox*

Элемент управления `ListBox` очень похож на комбинированный список, но применяется гораздо реже по двум причинам:

- в нем нельзя открыть список значений по кнопке. Все значения видны сразу в поле, аналогичном текстовому, и поэтому большое количество позиций в нем уместить трудно;
- пользователь не может вводить свои значения — только выбирать из готовых.

Пример этого элемента управления представлен на рис. 5.5.

Но у этого элемента управления есть и преимущества: в нем пользователь может выбирать не одно значение, как в `ComboBox`, а несколько.

Обычно `ListBox` используется:

- как промежуточное средство отображения введенных или выбранных пользователем через `ComboBox` значений (или любых других списков, например, для списка выбранных файлов);
- как средство редактирования списка значений, сформированных вышеуказанным образом или полученных из базы данных (для этого можно рядом с `ListBox` разместить кнопки **Удалить** или **Изменить**).

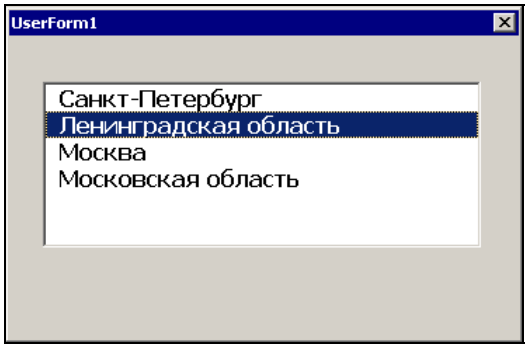


Рис. 5.5. Список (элемент управления `ListBox`) на форме

Основные свойства, методы и события у `ListBox` — те же, что и у `ComboBox`. Главное отличие — это свойство `MultiSelect`, которое позволяет пользователю выбирать несколько значений. По умолчанию это свойство отключено.

5.3.6. Элементы управления *CheckBox* и *ToggleButton*

Флажки (`CheckBox`) (пользователи часто называют их "галками" или "птичками") и *кнопки с фиксацией* (`ToggleButton`) используются для выбора невзаимоисключающих вариантов (если этих вариантов немного). Они представлены на рис. 5.6.

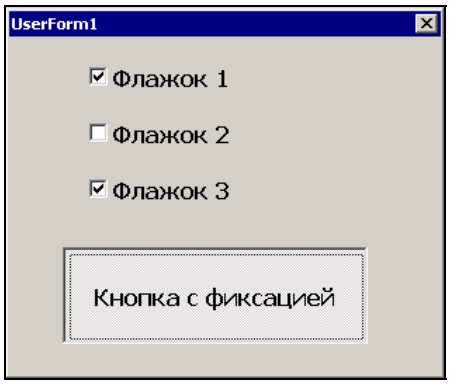


Рис. 5.6. Флажки (элементы управления `CheckBox`) и кнопки с фиксацией (`ToggleButton`)

Для `CheckBox` предусмотрено три главных свойства.

- `Caption` — надпись справа от флажка, которая объясняет, что выбирается этим флажком.

- TripleState — если это свойство установлено в False (по умолчанию), то флажок может принимать только два состояния: установлен или нет. Если для TripleState установить значение True, то появляется третье значение Null, когда установлен "серый" флажок. Такое значение часто используется, например, при выборе компонентов программы при установке, когда выбраны не все компоненты, а лишь некоторые.
- Value — само состояние флажка. Может принимать значения True (флажок установлен), False (снят) и Null — "серый" флажок (когда свойство TripleState установлено в True).

Главное событие элемента CheckBox — Change.

ToggleButton выглядит как кнопка, которая после щелчка на ней остается "нажатой" (рис. 5.6), а при повторном щелчке отключается. У нее могут быть те же два (или три, в соответствии со свойством TripleState) состояния, что и у CheckBox. Свойства и методы — те же самые. Единственное отличие — в восприятии их пользователем. Обычно ToggleButton воспринимается пользователем как переход в какой-то режим или начало выполнения продолжительного действия.

5.3.7. Элементы управления *OptionButton* и *Frame*

Если CheckBox предназначен для выбора невзаимоисключающих вариантов, то *переключатель* (OptionButton) используется как раз для выбора варианта в ситуации "или/или" (рис. 5.7).

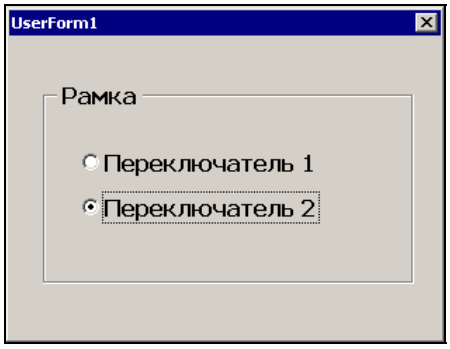


Рис. 5.7. Переключатели (2 объекта OptionButton) в рамке (объект Frame)

Классический пример, при помощи которого можно проиллюстрировать работу OptionButton, — выбор радиостанции на радиоприемнике: сразу две радиостанции слушать нельзя (поэтому иногда этот элемент управления называют RadioButton).

Главных свойств у этого элемента управления два.

- `Caption` — надпись для переключателя.
- `Value` — установлен переключатель или нет (только два состояния — `True` или `False`).

Главное событие тоже стандартное — `Change`.

Конечно, использовать один переключатель бессмысленно. Выбор должен предоставляться хотя бы из двух вариантов, и при выборе одного из них другой автоматически снимается. Однако в некоторых ситуациях нам необходимо выбрать из нескольких наборов вариантов (например, отчет за месяц/квартал/год, тип отчета, нужный филиал и т. п.). Решение простое — переключатели нужно сгруппировать.

Самый простой вариант группировки — просто использовать новую форму или вкладку на форме. Если переключатели находятся на одной форме (или на одной вкладке), они автоматически считаются взаимоисключающими. Если же нужно более точно выбрать группы, то необходимо использовать элемент управления `Frame`.

`Frame` — это просто рамка, которая выделяет прямоугольную область на форме и позволяет организовать элементы управления (рис. 5.7). Помещенные внутрь рамки переключатели считаются взаимоисключающими, остальные элементы управления ведут себя точно так же, хотя иногда бывает полезно с точки зрения наглядности свести вместе под одной рамкой, например, набор флажков. При желании рамку можно сделать невидимой, установив для свойства `BorderStyle` значение 1 и убрав значение свойства `Caption`.

5.3.8. Элемент управления *CommandButton*

Элемент управления `CommandButton` *(кнопка)* — самый распространенный элемент управления на формах (рис. 5.8).

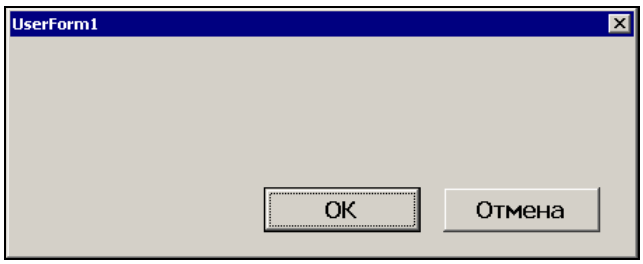


Рис. 5.8. Кнопки (объекты `CommandButton`)

В большинстве форм обязательно будет, по крайней мере, две кнопки: **OK** и **Отмена** (Cancel). По нажатию кнопки **OK** должно выполниться то действие,

ради чего создавалась форма, по нажатию кнопки **Отмена** форма должна закрыться. Ваша задача — обеспечить необходимый код для этих кнопок, который и будет выполнять эти действия.

Далее представлены самые важные свойства кнопки.

- `Cancel` — если для этого свойства установить значение `True`, то кнопка будет нажиматься автоматически при нажатии клавиши <Esc>. Как правило, на такие кнопки помещаются надписи типа "Отмена", "Выход", "Вернуться в окно приложения". Однако, кроме назначения клавише <Esc>, свойство ничего больше этой кнопке не дает. Необходимо еще добавить код в обработчик события `Click`, например:

```
Private Sub CommandButton1_Click()
    Unload Me
End Sub
```

  `Me` — это специальное зарезервированное слово, которое представляет текущий объект (в данном случае форму). Его можно использовать вместо имени формы.

- `Caption` — надпись, которая будет на кнопке.
- `Default` — если это свойство установлено в `True`, то такая кнопка будет считаться нажатой при нажатии пользователем клавиши <Enter>, даже если фокус находился в другом месте формы (но не на другой кнопке). Обычно такие кнопки являются главными, по которым выполняется действие, ради которого создавалась форма (печать отчета, занесение информации в базу данных, отправка почты и т. п.).
- `Picture` — если простая надпись вас не устраивает, можно назначить кнопке рисунок (пиктограмму).
- `TakeFocusOnClick` — определяет, будет ли передаваться управление этой кнопке при нажатии на нее. По умолчанию установлено `True`.

Главное событие для кнопки — это, конечно, `Click`. Как правило, к этому событию и привязывается программный код, ради которого создавалась кнопка.

5.3.9. Элементы управления *ScrollBar* и *SpinButton*

Полосы прокрутки (`ScrollBars`) чаще всего встречаются в текстовых полях, когда введенный текст полностью на экране не помещается. Однако ничего не мешает вам использовать `ScrollBar` как отдельный элемент управления (пользователи часто называют его "ползунок") для выбора какого-то значения из диапазона (рис. 5.9). Обычно такой элемент управления используется для выбора плавно меняющихся значений, например: уровня громкости, яркости, сжатия, приоритета и т. п.

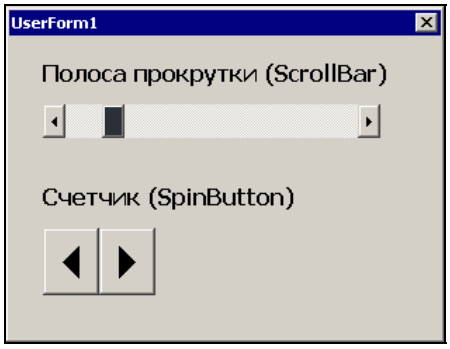


Рис. 5.9. Полоса прокрутки (`ScrollBar`) и счетчик (`SpinButton`)

Главное событие для `ScrollBar` — уже знакомое нам `Change`. Главные свойства этого элемента управления представлены далее.

- `Max` и `Min` — максимальное и минимальное значения, которые можно задать при помощи этого элемента управления. Возможный диапазон — от –32 767 до +32 767. При этом максимальное значение может быть и меньше минимального — просто ползунок придется тянуть в обратную сторону.
- `LargeChange` и `SmallChange` — определяют, какими шагами будет двигаться ползунок при перемещении его пользователем (путем щелчка на полосе около ползунка или при нажатии на одну из кнопок направления соответственно).
- `Orientation` — определяет расположение ползунка (вертикальное или горизонтальное). По умолчанию для этого свойства установлено значение 1, т. е. ориентация определяется автоматически в зависимости от конфигурации отведенного элементу управления пространства на форме (что больше — длина или высота). Однако при помощи этого свойства можно и явно указать вертикальное или горизонтальное расположение ползунка.
- `ProportionalThumb` — определяет размер ползунка: будет ли он пропорционален размеру полосы прокрутки (по умолчанию) или будет фиксированного размера.
- `Value` — главное свойство этого элемента управления. Определяет положение ползунка и то значение, которое будет возвращать этот элемент управления программе.

Как правило, использование ползунка без отображения выбранной при помощи его информации не очень приветствуется пользователями. В самом простом варианте то, что выбрано при помощи ползунка, следует просто отображать в текстовой надписи:

```
Private Sub ScrollBar1_Change()
    Label1.Caption = ScrollBar1.Value
End Sub
```

В более сложном варианте пользователю можно выбирать — использовать ли ползунок или вводить значение в текстовом поле. В этом случае в событии `Change` для текстового поля необходимо предусмотреть проверку вводимых пользователем значений и обратную связь с ползунком.

Элемент управления *счетчик* (`SpinButton`) — эта та же полоса прокрутки, лишенная самой полосы и ползунка (рис. 5.9). `SpinButton` используется в тех ситуациях, когда диапазон выбираемых значений совсем небольшой (например, надо выбрать количество копий для печати отчета). Все свойства, которые есть у `SpinButton`, совпадают со свойствами `ScrollBar`.

5.3.10. Элементы управления *TabStrip* и *MultiPage*

Набор вкладок (`TabStrip`) и *набор страниц* (`MultiPage`) применяются в одной и той же ситуации — когда элементов управления слишком много, чтобы уместить их на одной странице формы. Эти элементы управления позволяют создавать на форме несколько вкладок (страниц), между которыми сможет переходить пользователь. Принципиальное отличие между этими элементами управления заключается в том, что на вкладках `TabStrip` всегда располагаются одинаковые элементы управления, а на `MultiPage` — разные. Применение множества вкладок вы наверняка видели во многих программах (например, в Word в окне **Параметры**, открывающемся с помощью меню **Сервис | Параметры**). Пример использования элемента управления `MultiPage` представлен на рис. 5.10.

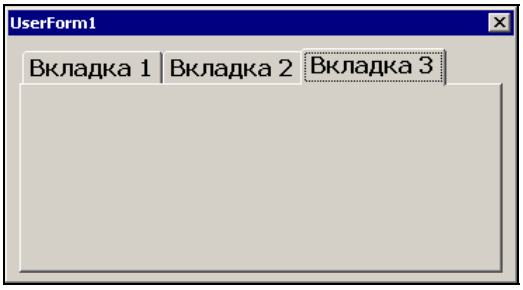


Рис. 5.10. Форма с несколькими вкладками (элементами управления `MultiPage`)

Элемент `TabStrip` используется реже. Например, его можно применить для занесения данных по одному шаблону для филиалов или сотрудников (если их не слишком много).

Свойства и события у этих элементов управления практически идентичны. Чаще всего используются следующие свойства.

- `MultiRow` — определяет, можно ли использовать несколько горизонтальных рядов вкладок.
- `TabOrientation` — определяет, где будут расположены заголовки вкладок (по умолчанию вверху).
- `Value` — номер вкладки, которая открыта в настоящий момент (нумерация начинается с 0).

Главное событие этих элементов управления — `Change` (т. е. переход между вкладками). К нему можно привязать, например, проверку уже введенных пользователем значений или вывод предупреждений.

5.3.11. Элемент управления *Image*

Наверное, *рисунок* (`Image`) — это самый простой из элементов управления. Он позволяет отобразить на форме рисунок в одном из распространенных форматов, который будет реагировать на щелчок мышью (а может просто использоваться для украшения формы). Отметим некоторые моменты, связанные с применением элемента управления `Image`:

- в качестве альтернативы можно использовать свойство `Picture` формы (особенно если вам нужен фоновый рисунок для всей формы);
- еще две альтернативы — это свойство `Picture` элементов управления `Label` или `CommandButton`. Функциональность рисунков получается практически одинаковая;
- при использовании этого элемента управления само изображение копируется внутрь документа и внешний его файл больше не нужен.

Главные свойства этого элемента управления представлены далее.

- `Picture` — позволяет выбрать само изображение для формы.
- `PictureAlignment` — позволяет выбрать местонахождение изображения в отведенной ему области. По умолчанию рисунок располагается по центру.
- `PictureSizeMode` — позволяет выбрать режим растяжения или уменьшения элемента в случае, если он не соответствует размеру области.
- `PictureTiling` — определяет, размножать ли маленький рисунок, чтобы он покрыл всю отведенную ему область ("замостить").

Главное событие элемента управления `Image` — `Click`.

5.3.12. Применение дополнительных элементов управления

Мы рассмотрели стандартные элементы управления, которые изначально помещены на панель **ToolBox** и доступны для размещения в формах. Однако возможности форм VBA этим не ограничиваются. В вашем распоряжении — сотни и тысячи элементов управления, встроенных в Windows, в другие продукты или поставляемые отдельно (в том числе третьими фирмами). Для того чтобы можно было разместить их на форме, щелкните правой кнопкой мыши по пустому пространству в **ToolBox**, выберите пункт контекстного меню **Additional Controls**, а затем в списке выберите нужный элемент. Правда, при использовании нестандартных элементов управления необходимо помнить, что при переносе программы (файла Office) на другой компьютер вам потребуется обеспечить на нем наличие необходимых библиотек.

Очень часто в программах используются дополнительные элементы управления Internet Explorer, Acrobat Reader, календарь, проигрыватели аудио- и видеофайлов и т. п. Например, чтобы разместить на форме элемент управления Microsoft Web Browser (в русифицированной версии Windows он называется **Обозреватель веб-страниц (Microsoft)**), который представляет окно Internet Explorer, нужно выполнить следующие действия:

- щелкнуть правой кнопкой мыши по пустому пространству в окне **Toolbox** и в контекстном меню выбрать **Additional Controls**;
- в открывшемся списке выбрать **Microsoft Web Browser** (или **Обозреватель веб-страниц (Microsoft)**);
- изменившимся курсором мыши очертить на форме ту область, которую будет занимать этот элемент управления.

Далее нужно позаботиться о программном коде для этого элемента управления. Созданный нами на форме элемент управления по умолчанию будет называться `WebBrowser1`. Можно выбрать любое из доступных событий этого элемента управления, а можно использовать этот элемент управления и в событиях других объектов. Например, если нам нужно, чтобы при открытии формы в окне Internet Explorer на ней открывалась определенная страница, можно воспользоваться событием `Initialize` для формы. Соответствующий код может быть таким:

```
Private Sub UserForm_Initialize()
    WebBrowser1.Navigate "http://www.AskIt.ru"
End Sub
```

Преимущества использования этого элемента управления очевидны — вы можете расширить функциональность своей формы за счет использования

Web-страниц (например, с формами HTML). Internet Explorer обычно установлен на любом компьютере под управлением Windows и поэтому с этим элементом управления не возникает никаких проблем при переносе программы на другой компьютер. Справку по этому элементу управления придется смотреть в MSDN.

Еще один часто используемый элемент управления, который есть практически на всех компьютерах — `Calendar` *(календарь)* (рис. 5.11). В зависимости от версии вашей операционной системы и установленного программного обеспечения он может называться по-разному, у меня на компьютере он называется **Calendar Control 8.0**. При помощи этого элемента управления пользователю будет очень удобно выбирать нужную дату.

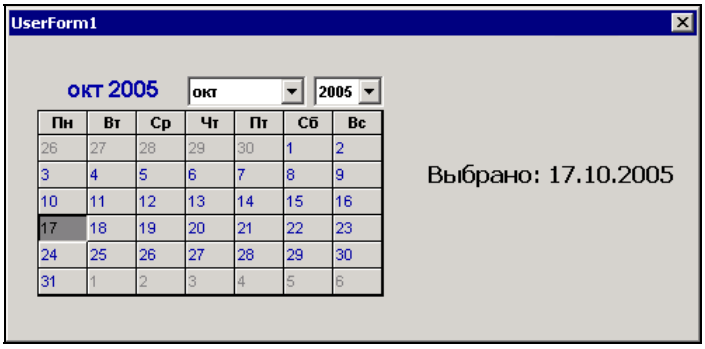


Рис. 5.11. Элемент управления `Calendar` и надпись, дублирующая значение, выбранное пользователем в `Calendar`

Главное свойство этого элемента управления — `Value`, т. е. та дата, которая выбрана пользователем. Остальные свойства предназначены для отображения внешнего вида календаря.

В Excel на панели **ToolBox** имеется еще один специфический элемент управления — `RefEdit` (в списке **Additional Controls** он называется как **RefEdit.Ctrl**). Он похож на текстовое поле с кнопкой справа. При нажатии на эту кнопку форма, на которой размещен этот элемент управления, "спрячется", а пользователю будет предоставлена возможность выбрать одну ячейку или диапазон ячеек Excel. После того как пользователь завершит выбор, он опять вернется в окно формы, а в `RefEdit` будет помещена информация об адресе выбранного диапазона. Такой же адрес, конечно, можно вводить и вручную. Главное свойство этого элемента управления — `Value`.

Большое количество дополнительных элементов управления предусмотрено для форм Access. Они являются специфическими для Access, и про них будет рассказано в *гл. 12*.

Задание для самостоятельной работы 5: Работа с элементами управления

Подготовка:

1. Создайте новую книгу Excel и сохраните ее как Prikaz.xls. Заполните ячейки с A1 по A5 значениями, аналогичными представленным на рис. 5.12. Данные о сотрудниках лучше ввести в родительном падеже, поскольку эти значения будут подставляться в автоматически создаваемый приказ в формате документа Word.

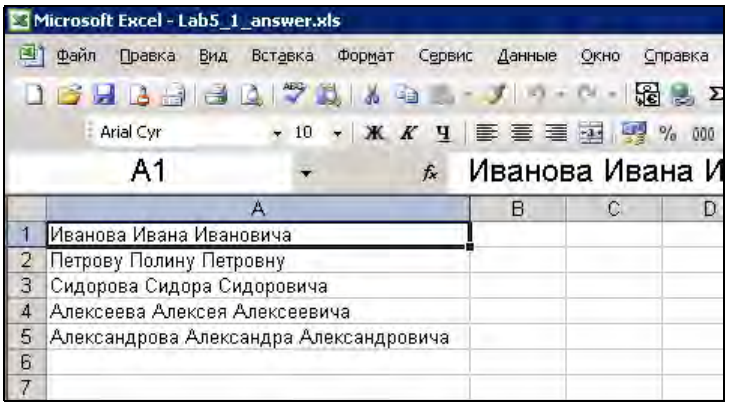


Рис. 5.12. Список сотрудников на листе Excel

2. Откройте редактор Visual Basic и в окне **Project Explorer** щелкните правой кнопкой мыши по объекту **Эта книга** и в контекстном меню выберите **View Code**.
3. В окне редактора кода для этой книги введите следующий код:

```
'При открытии рабочей книги показываем форму UF1
Private Sub Workbook_Open()
    UF1.Show
End Sub

'Специальная процедура, которая печатает приказ в Word
Public Sub DocWrite(sPovod As String, sFio As String, bFlagPremia As
Boolean, bFlagGramota As Boolean, nSummaPremii As Long, sOtvIsp As
String)

    Dim oWord As Word.Application
    Dim oDoc As Word.Document
    Set oWord = CreateObject("Word.Application")
```

```
    Set oDoc = oWord.Documents.Add()
    oWord.Visible = True
    oDoc.Activate
    With oWord.Selection
        .TypeText "Приказ"
        .Style = "Заголовок 1"
        .ParagraphFormat.Alignment = wdAlignParagraphCenter
        .TypeText vbCrLf
        .Style = "Обычный"
        .TypeText vbCrLf
        .TypeText "г.Санкт-Петербург" & Space(90) & Date
        .TypeText vbCrLf
        .TypeText vbCrLf
        .TypeText "За проявленные успехи в " & sPovod & _
                  " наградить " & sFio & ":"
        .TypeText vbCrLf
        If bFlagPremia Then
            .TypeText vbTab & "- денежной премией в сумме " & _
                      nSummaPremii & " рублей"

        End If
        If bFlagGramota Then
            .TypeText vbCrLf
            .TypeText vbTab & "- почетной грамотой."
        Else
            .TypeText "."
        End If
        .TypeText vbCrLf
        .TypeText vbCrLf
        .TypeText vbCrLf
        .TypeText vbCrLf
        .TypeText "Генеральный директор" & vbTab & vbTab & _
                  vbTab & "Иванов И. И."
        .ParagraphFormat.Alignment = wdAlignParagraphCenter
        .TypeParagraph
        .TypeText vbCrLf
        .TypeText vbCrLf
        .ParagraphFormat.Alignment = wdAlignParagraphLeft
        .TypeText Text:=("Отв. исполнитель " & sOtvIsp)
        .TypeParagraph

    End With
End Sub
```

4. В окне **Project Explorer** щелкните правой кнопкой мыши по проекту Prikaz.xls и в контекстном меню выберите **Insert | UserForm**. Выделите созданный вами объект формы и нажмите клавишу <F4>. В окне **Properties** введите для свойства `Name` этой формы значение `UF1`.

   Поместите на форму из **Toolbox** единственную кнопку — элемент управления `CommandButton1`. Установите для этой кнопки значение свойства `Caption` как `"Напечатать приказ"` (без кавычек) и измените размеры и местонахождение этой кнопки, чтобы форма выглядела так, как показано на рис. 5.13.

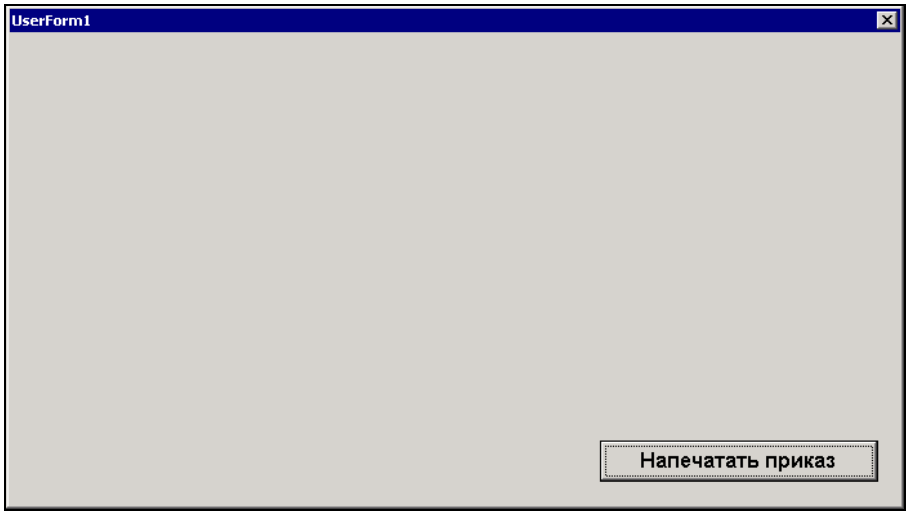


Рис. 5.13. Заготовка для формы с единственной кнопкой

5. Щелкните правой кнопкой мыши по кнопке `CommandButton1` на вашей форме, в контекстном меню выберите **View Code** и добавьте в код событийной процедуры для события `Click` этой кнопки следующий код:

```
Private Sub CommandButton1_Click()
    Dim sPovod As String
    Dim sFio As String
    Dim bFlagPremia As Boolean
    Dim bFlagGramota As Boolean
    Dim nSummaPremii As Long
    Dim sOtvIsp As String

    'Подставить данные из формы
    sPovod = "освоении новых информационных технологий"
    sFio = "Иванова Ивана Ивановича"
```

```
    bFlagPremia = True
    bFlagGramota = True
    nSummaPremii = 100000
    sOtvIsp = "Петрова П. П."
    'Конец подстановки данных

    Call ЭтаКнига.DocWrite(sPovod, sFio, bFlagPremia, bFlagGramota, _
        nSummaPremii, sOtvIsp)
End Sub
```

6. Запустите вашу форму на выполнение и убедитесь, что она работает: выводит в создаваемый документ Word приказ с фиксированными значениями.

ЗАДАНИЕ:

Измените форму таким образом, чтобы вместо присвоения переменным в выделенном комментариями блоке заранее определенных значений пользователь мог выбирать данные при помощи формы. При этом:

1. Значение переменной `sPovod` должно выбираться из трех возможных значений: "освоении новых информационных технологий", "внедрении новых программных продуктов" и значение, которое пользователь может ввести через текстовое поле. Используйте для этого набор из трех переключателей и текстовое поле (оно должно быть спрятано, если пользователь выбрал один из первых двух переключателей). По умолчанию должно подставляться "освоении новых информационных технологий".
2. Значение переменной `sFio` должно выбираться пользователем при помощи комбинированного списка. В этот комбинированный список должны автоматически помещаться значения из всех непустых ячеек столбца А листа Excel. По умолчанию должно выбираться значение "Иванова Ивана Ивановича".

Примечание

Образец для работы с ячейками столбца можно получить из ответов к предыдущим лабораторным работам.

3. Значения переменных `bFlagPremia` и `bFlagGramota` должны устанавливаться в зависимости от состояния двух флажков — "Премия" и "Грамота". По умолчанию оба флажка должны быть установлены. Если пользователь снял оба флажка, то ему должно выводиться предупреждающее сообщение "Не выбрана ни премия, ни почетная грамота!" с отменой вывода документа.
4. Пользователь должен иметь возможность задавать значение переменной `nSummaPremii` либо при помощи полосы прокрутки с диапазоном значений

от 0 руб. до 100 000 руб., либо при помощи текстового поля. Если флажок "Премия" снят, то полоса прокрутки и текстовое поле должны быть спрятаны от пользователя.

Ход полосы прокрутки (увеличение или уменьшение значения при щелчке на кнопках со стрелками) должен быть равен 100 руб.

По умолчанию размер премии должен быть равен 100 руб.

5. Поместите на форму еще одну кнопку **Отмена**. Эта кнопка должна закрывать текущую форму и срабатывать при нажатии клавиши <Esc>.
6. В заголовке формы должно выводиться значение "Формирование приказа о выплате премии".

Общий вид формы может выглядеть, например, так, как представлено на рис. 5.14.

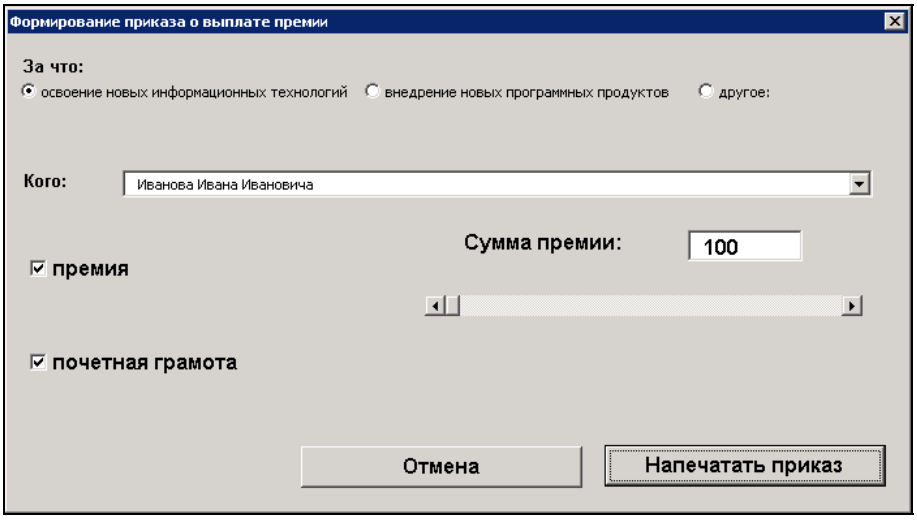


Рис. 5.14. Готовая форма

Ответ к заданию 5

К пункту 1 задания (работа с переключателями и текстовым полем):

1. В окне **Project Explorer** два раза щелкните мышью по объекту формы `UF1`. Затем в **ToolBox** щелкните по объекту `Label` и отведите место этому элементу управления в верхней части формы. Щелкните правой кнопкой мыши по созданному элементу управления `Label1` и в контекстном меню выберите **Properties**. Измените значение свойства `Caption` на `"За что:"` и при помощи свойства `Font` подберите подходящий шрифт и его размер.

2. В **ToolBox** щелкните по элементу управления `OptionButton` и отведите на форме место этому элементу управления. Повторите эту операцию еще два раза.
3. Откройте свойства первого переключателя. Измените значение свойства `Name` на `optOsvoenie`, а значение свойства `Caption` — на `"освоение новых информационных технологий"`. Для второго переключателя поменяйте значение свойства `Name` на `optVnedrenie` и свойство `Caption` — на `"внедрение новых программных продуктов"`, для третьего — на `optDrugoe` и `"другое:"` соответственно.
4. В **ToolBox** щелкните по элементу управления `TextBox` и поместите его в нужное место формы. Установите для свойства `Name` этого элемента управления значение `txtDrugoe`.
5. Щелкните правой кнопкой мыши по пустому месту на форме и в контекстном меню выберите **View Code**. В списке событий в верхней части окна редактора кода выберите событие `Initialize` для `UserForm` и введите для него следующий код:

```
optOsvoenie.Value = True
txtDrugoe.Visible = False
```

6. Для события `Change` переключателя `optDrugoe` введите следующий код:

```
If optDrugoe.Value = True Then
    txtDrugoe.Visible = True
Else
    txtDrugoe.Visible = False
End If
```

7. Перейдите к коду события `Click` для `CommandButton1` и вместо строки:

```
sPovod = "освоении новых информационных технологий"
```

   введите следующий код:

```
If optOsvoenie.Value = True Then sPovod = _
    "освоении новых информационных технологий"
If optVnedrenie.Value = True Then sPovod = _
    "внедрении новых программных продуктов"
If optDrugoe.Value = True Then sPovod = txtDrugoe.Value
```

8. Запустите форму на выполнение, напечатайте приказ и убедитесь, что все работает согласно поставленным условиям.

К пункту 2 задания (работа с комбинированным списком):

1. Разместите на форме еще один элемент управления `Label` с надписью `"Кого:"` и настройте для него шрифт.

2. Щелкните в **Toolbox** по элементу управления ComboBox и выделите для него место на форме. Присвойте созданному элементу управления ComboBox имя cbFIO.

3. Откройте код для события Initialize нашей формы UserForm и дополните его следующими строками:

```
Dim oColumn As Range
Dim oCell As Range
Set oColumn = Columns("A")
For Each oCell In oColumn.Cells
    If oCell.Value <> "" Then
        cbFIO.AddItem oCell.Value
    End If
Next
cbFIO.ListIndex = 0
```

4. Перейдите к коду события Click для CommandButton1 и вместо строки:

```
sFio = "Иванова Ивана Ивановича"
```

введите следующий код:

```
sFio = cbFIO.Value
```

5. Запустите форму на выполнение и убедитесь, что все работает нормально.

К пункту 3 задания (работа с флажками):

1. При помощи **ToolBox** поместите на форму два элемента управления CheckBox. Для первого элемента свойству Name присвойте значение chPremia и для свойства Caption — значение "Премия", для второго — chGramota и "Почетная грамота" соответственно.

2. Откройте код события Initialize формы UserForm и дополните его следующими строками:

```
chPremia.Value = True
chGramota.Value = True
```

3. Перейдите к коду события Click для CommandButton1 и вместо строк:

```
bFlagPremia = True
bFlagGramota = True
```

введите следующий код:

```
bFlagPremia = chPremia.Value
bFlagGramota = chGramota.Value
```

```
If bFlagPremia = False And bFlagGramota = False Then
    MsgBox "Не выбрана ни премия, ни почетная грамота!"
    Exit Sub
End If
```

4. Запустите форму на выполнение и убедитесь, что все работает нормально.

К пункту 4 задания (применение полосы прокрутки и дублирующего текстового поля):

1. Поместите на форму еще один элемент управления `Label` с надписью `"Сумма премии:"`. Присвойте его свойству `Name` значение `lblSum`.
2. Поместите рядом текстовое поле и присвойте его свойству `Name` значение `txtSum`.
3. Разместите под текстовым полем элемент управления `ScrollBar` и присвойте следующие значения его свойствам:
   - `Name` — значение `sbSum`;
   - `Min` — значение 0;
   - `Max` — значение 100 000;
   - `SmallChange` — значение 100.
4. Для события `Change` элемента управления `sbSum` введите следующий код:

```
txtSum.Value = sbSum.Value
```

5. Для события `Change` элемента управления `txtSum` введите следующий код:

```
sbSum.Value = CLng(txtSum.Value)
```

Примечание

Такой код является потенциально опасным, поскольку не проверяется вводимое пользователем в текстовом поле значение. Если это значение будет невозможно преобразовать в числовое или оно окажется больше 100 000, то возникнет ошибка времени выполнения. Как предупреждать появление ошибок и перехватывать их, будет рассмотрено в *гл. 6*.

6. Для события `Initialize` нашей формы `UserForm` добавьте следующий код:

```
sbSum.Value = 100
txtSum.Value = 100
```

7. Для события `Change` элемента управления `chPremia` добавьте следующий код:

```
If chPremia.Value = False Then
    lblSum.Visible = False
```

```
    txtSum.Visible = False
    sbSum.Visible = False
Else
    lblSum.Visible = True
    txtSum.Visible = True
    sbSum.Visible = True
End If
```

8. Для кода `Click` кнопки `CommandButton1` вместо кода:

   ```
   nSummaPremii = 100000
   ```

   впишите код:

   ```
   nSummaPremii = sbSum.Value
   ```

9. Запустите форму на выполнение и убедитесь, что все работает нормально.

К пункту 5 задания (применение кнопки):

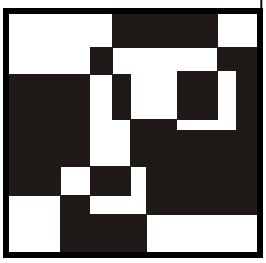
1. Разместите на поле еще одну кнопку и настройте значения ее свойств следующим образом:
   - `Name` — значение `btnEscape`;
   - `Caption` — значение "Отмена";
   - `Cancel` — значение `True`.
2. Для события `Click` этой кнопки поместите код

   ```
   Unload Me
   ```

К пункту 6 задания (изменение заголовка формы):

1. Щелкните правой кнопкой мыши по пустому месту на форме и в контекстном меню выберите **Properties**.
2. Для свойства `Caption` настройте значение "`Формирование приказа о выплате премии`".
3. Запустите форму на выполнение и убедитесь, что приказы печатаются правильно.

ГЛАВА 6



Отладка и обработка ошибок в программе

6.1. Типы ошибок

Если вы выполняли все задания для самостоятельной работы, то, наверное, уже заметили, что при написании программного кода допустить ошибку очень просто. Одна из задач разработчика — найти такие ошибки и устранить их (или обеспечить перехват ошибок времени выполнения и нормальную работу приложения даже в случае возникновения этих ошибок).

Все ошибки можно разделить на три большие группы:

- *синтаксические* (неправильно написан оператор, имя переменной и т. п.). Такие ошибки не требуют больших усилий по их поиску и исправлению. Многие синтаксические ошибки "отлавливаются" редактором кода VBA еще в процессе ввода кода. Об обнаружении других ошибок сообщается в ходе компиляции и запуска программы. При этом компилятор VBA выдает информацию о том, в какой строке кода обнаружена ошибка и в чем она заключается. Рекомендуется проверить данную строку по справке VBA;
- *логические*. В ходе выполнения программа ведет себя не так, как вы планировали. Главное здесь — найти причину неправильного поведения программы. Обычно для выявления и исправления ошибок такого типа предназначены приемы отладки *(см. разд. 6.2)*;
- *ошибки времени выполнения* (run-time error). Они возникают, когда в процессе выполнения программа столкнулась с проблемой, решить которую она не в состоянии (файл с таким именем уже существует, возник конфликт записей при вставке в базу данных, произведена попытка записать информацию на переполненный диск и т. п.). Заранее предугадать, какая именно неприятность может случиться, очень сложно. Во многом квалификация программиста определяется тем, как он умеет предугадывать

возможности возникновения ошибок времени выполнения и обеспечивать их перехват и обработку.

Если программа делается "для себя" (для автоматизации работы того пользователя, который пишет эту программу), то очень часто перехват ошибок времени выполнения вообще не предусматривается. Возникла ошибка — ничего страшного: открыли программу в отладчике, посмотрели, отчего возникла ошибка, и "исправились". Но если программа пишется для передачи другим пользователям (особенно не очень квалифицированным), то на реализацию обработки ошибок времени выполнения обычно уходит больше времени, чем на создание самой логики программы.

6.2. Приемы отладки. Окна *Immediate*, *Locals* и *Watch*

6.2.1. Тестирование

Главный способ обеспечения безошибочной работы программы — это ее тестирование. При создании крупных программных продуктов на их тестирование часто уходит не меньше времени, чем на создание. Поскольку в наших условиях рассчитывать на то, что тестировать вашу программу будет профессиональный тестер, не приходится, проверять ее придется вам самим. Приведу некоторые советы по тестированию:

- попытайтесь запустить программу при работе с большим количеством документов или когда не открыто ни одного документа;
- посмотрите, как работает программа, когда окно документа развернуто, свернуто или размер его изменен;
- проверьте, как работает программа, когда выделены разные элементы или группы элементов;
- если предусматривается ввод информации, попробуйте специально передать программе неверные значения. Например, если программа ожидает числовых значений, попробуйте ввести строковое значение, значение даты или оставить поле пустым;
- попробуйте прервать работу программы в самый неподходящий момент и потом вновь запустить ее;
- проверьте, как ведет себя программа, когда пропадает сеть, заканчивается свободное место на диске, заканчивается бумага в принтере и т. п.;
- проверьте работу программы под разными версиями Office и операционных систем (в том числе англоязычных и локализованных);

- попробуйте до запуска программы и во время ее работы переставлять системную дату и время, устанавливая самые невероятные значения.

Если есть возможность, всегда рекомендуется немного поработать, выполняя обязанности пользователя, для которого создается программа.

Мне очень нравится "диверсионный" подход при тестировании программ. Представьте себе, что вы — вредитель и диверсант, у которого цель — вывести программу из строя. Потом опробуйте те способы, которые вам пришли в голову. Если способ оказался удачным, придумайте для него защиту. Как ни удивительно, но реальная работа пользователей с вашей программой будет очень похожа на действия таких диверсантов.

6.2.2. Переход в режим паузы

Один из самых важных приемов в ходе отладки программы — возможность вовремя остановиться в ходе выполнения, чтобы просмотреть значения переменных, вмешаться в ход выполнения программы вручную, просмотреть, что возвращает оператор или функция и т. п.

Программу в режим паузы можно перевести следующими способами:

- с самого начала запустить программу в режиме пошагового выполнения (меню **Debug | Step Into** или клавиша <F8>). В этом случае программа будет переходить в режим паузы после выполнения каждого оператора;
- установить в программе точку останова *(breakpoint)*. Это можно сделать, поставив указатель на нужной строке и в меню **Debug** выбрав **Toggle Breakpoint** (или нажав клавишу <F9>). Строка с точкой останова будет помечена коричневым цветом, и точка такого же цвета появится на рамке слева от строки. Второй вариант — просто щелкнуть мышью по рамке слева от строки. Снятие точки останова — повторить то же самое действие еще раз. При запуске программа автоматически остановится на первой точке останова;
- к сожалению, точки останова не сохраняются после закрытия документа. Если нужно запомнить место остановки между сеансами отладки, то нужно просто впечатать в это место строку с единственной командой `Stop`. Программа в ходе выполнения автоматически остановится на этой строке, например:

```
n1 = 10
n2 = 5
Stop
nResult = n1/n2
```

- если программа не хочет завершаться (например, у вас выполняется бесконечный цикл), в ходе ее выполнения можно нажать кнопку **Break** на па-

нели инструментов **Standard**, воспользоваться меню **Run | Break** или просто нажать клавиши <Ctrl>+<Break>;

- еще одна возможность приостановить выполнение программы — воспользоваться контролируемым выражением (в окне **Watches**). Об этом — в *разд. 6.2.6*.

В любом случае выполнение будет приостановлено в выбранном вами месте программы, и следующий оператор, который должен быть выполнен, будет выделен желтым цветом (рис. 6.1).

Что делать дальше, рассказано в следующем разделе.

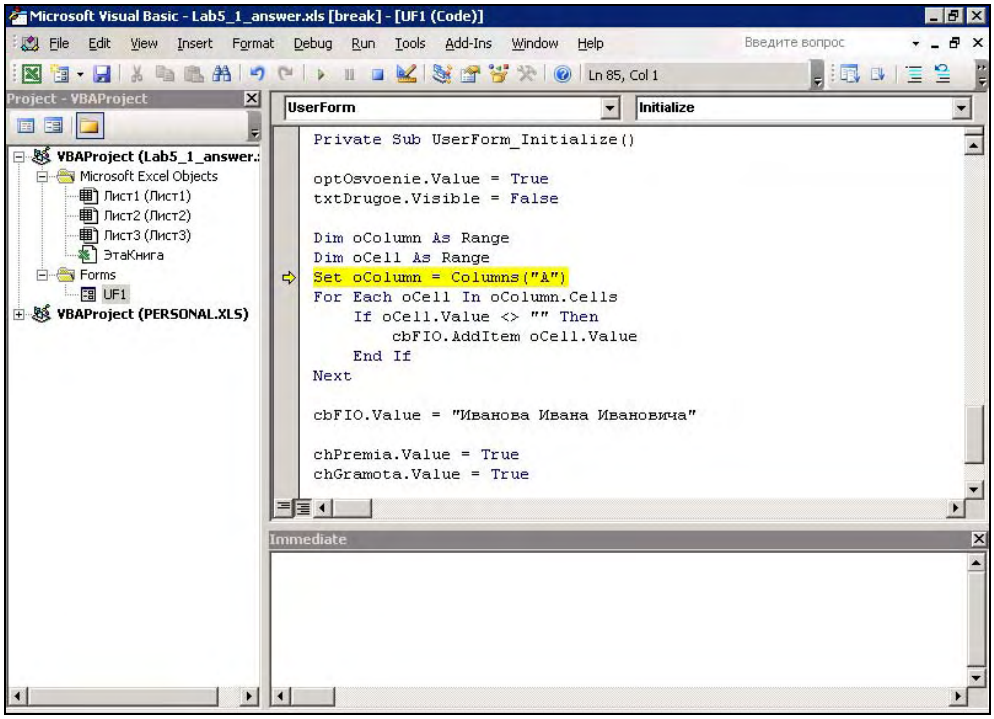


Рис. 6.1. Выполнение программы приостановлено

6.2.3. Действия в режиме паузы

В режим паузы обычно входят для того, чтобы предпринять какие-то действия. В этом режиме можно сделать следующее:

- продолжить выполнение в пошаговом режиме. Для этого можно воспользоваться меню **Debug | Step Into** или нажать клавишу <F8>. При этом бу-

дет выполнена текущая строка (помеченная желтым цветом), и выделится следующая строка вашего программного кода. Если была выделена строка, в которой происходит вызов процедуры или функции, то мы "шагнем" *(Step Into)* внутрь этой процедуры;

- если в строке программы происходит вызов какой-то процедуры, которая уже отлажена и проблем вызвать не должна, то у вас есть возможность выполнить ее без остановок и перейти к следующему оператору. Для этой цели используется команда меню **Debug | Step Over** (или <Shift>+<F8>);
- если вы все-таки зашли в процедуру, выполнили там необходимые действия и хотите быстро довести ее выполнение до конца, то в вашем распоряжении — меню **Debug | Step Out** (или <Ctrl>+<Shift>+<F8>);
- если вы хотите исполнить код не пошагово, а фрагментами (например, чтобы не проводить пошаговое выполнение больших циклов), вы можете щелкнуть правой кнопкой мыши по нужной строке кода и в контекстном меню выбрать **Run to Cursor** (альтернатива — воспользоваться той же командой в меню **Debug** или нажать <Ctrl>+<F8>). Программа пройдет вперед на выбранную вами строку и вновь остановится;
- если вам нужно "перепрыгнуть" через какой-то фрагмент кода, вызывающий ошибку (т. е. просто пропустить его без выполнения), можно воспользоваться командой меню **Debug | Set Next Statement** (или нажать <Ctrl>+<F9>), а затем перетащить желтую стрелку слева от кода вниз (или вверх) на нужную строку. В последних версиях Office в режиме останова можно сразу перетаскивать желтую стрелку, не вызывая команды. Альтернативный способ пропуска команд — выделить ненужный блок и при помощи кнопки **Comment Block** панели инструментов **Edit** его закомментировать (но тогда потом придется снимать комментарии);
- если в процессе просмотра кода вы пролистали код достаточно далеко и вам хочется вернуться к месту остановки (строка, выделенная желтым) без долгих поисков, то в вашем распоряжении команда **Show Next Statement**;
- чтобы просто продолжить выполнение программы после остановки, можно нажать клавишу <F5> или воспользоваться командой меню в меню **Run | Continue** (она появится вместо команды **Run**). Прекратить выполнение программы можно при помощи команды **Reset** в том же меню или клавишами <Alt>+<F4>. Кроме того, в вашем распоряжении одноименные кнопки на панели инструментов **Standard**.

Чаще всего переход в режим остановки нужен, чтобы просмотреть текущие значения переменных или исправить код. Текущие значения переменных можно просмотреть при помощи окон **Immediate**, **Locals** или **Watch** (о них будет рассказано в следующих разделах), а можно воспользоваться подсказ-

ками в самом коде. Для того чтобы получить информацию о текущем значении переменной, достаточно навести на нее указатель мыши — рис. 6.2 (если у вас остался включен по умолчанию параметр **Auto Data Tips** в окне **Options**, меню **Tools | Options**).

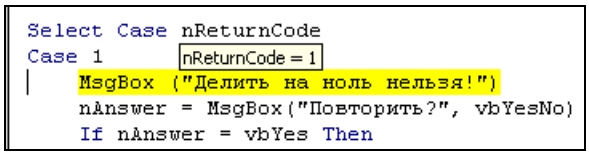


Рис. 6.2. Просмотр текущих значений переменных

Чтобы просмотреть информацию об области видимости и типе данных переменной, необходимо установить на нее курсор ввода текста и в меню **Edit** выбрать **Quick Info** (можно нажать клавиши <Ctrl>+<I>).

Все возможности редактирования кода в режиме отладки остаются. Если вы поняли, что переменной (свойству объекта) присвоено неверное значение, то вы можете исправить соответствующую строку, вернуться назад при помощи **Set Next Statement** (<Ctrl>+<F9>) и продолжить выполнение с исправленным значением.

Но возможностей подсказок в коде нам не всегда достаточно. Когда переменных очень много, или, например, к проблемам может привести значение свойства, которое мы даже не назначали, или во многих других ситуациях нам могут потребоваться специализированные окна **Immediate**, **Locals** и **Watch**.

6.2.4. Окно *Immediate*

Это мое любимое средство отладки. Окно **Immediate** предназначено для немедленного *(immediate)* выполнения программного кода, вызвать его можно через меню **View** или клавишами <Ctrl>+<G>. В этом окне можно:

- просматривать и изменять значения переменных и свойств объекта. Посмотреть значения переменных можно, набрав в окне **Immediate**:

```
Print nResult
Print oDoc.FullName
```

или еще проще:

```
?nResult
?oDoc.FullName
```

В данном случае `Print()` — это метод объекта `Debug`. Вывод в окно **Immediate** можно произвести при помощи этого объекта и просто из кода программы:

```
Debug.Print nResult
```

Преимуществом этого метода перед обычным `MsgBox()` является то, что при работе не в отладочном окружении (т. е. когда ваша программа уже эксплуатируется пользователями) все вызовы методов объекта `Debug` просто игнорируются (у этого объекта есть еще один метод `Assert()` — переход по условию).

Изменение значений переменных и свойств в окне **Immediate** производится точно так же, как в коде программы;

- вызывать процедуры и функции вашей программы или методы объектов — точно так же, как в коде программы. Microsoft рекомендует перед вставкой в программу проверять потенциально опасный код (например, который может привести к зависанию системы) в этом окне;
- а можно использовать это окно просто как калькулятор, вводя там выражения вида:

```
Print 25*115
```

Пример окна **Immediate** с некоторыми выполненными действиями представлен на рис. 6.3.

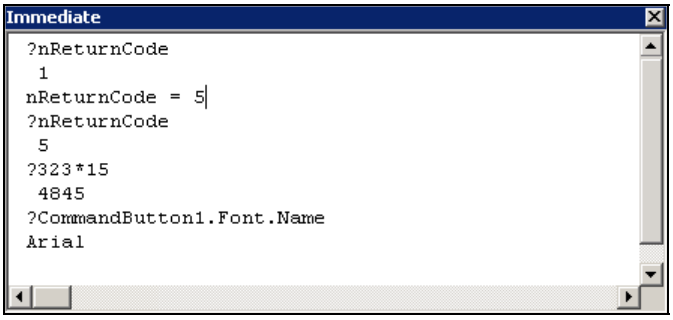


Рис. 6.3. Пример использования окна **Immediate**

Чтобы не печатать в окне **Immediate** выражения и имена переменных, которые уже есть в коде, можно воспользоваться перетаскиванием их в окно **Immediate** из окна редактора кода с нажатой клавишей <Ctrl> (чтобы происходило копирование).

6.2.5. Окно *Locals*

Очень часто бывает так, что вам нужно просмотреть значения всех переменных и свойств объектов, чтобы определить недопустимые, и сразу же их поменять. В этом случае возиться с каждым свойством/переменной в окне **Immediate** неэффективно. В этой ситуации гораздо удобнее использовать окно **Locals** (меню **View | Locals Window**). В этом окне выводятся значения всех переменных и свойств объектов, доступных в настоящий момент (пример окна приведен на рис. 6.4).

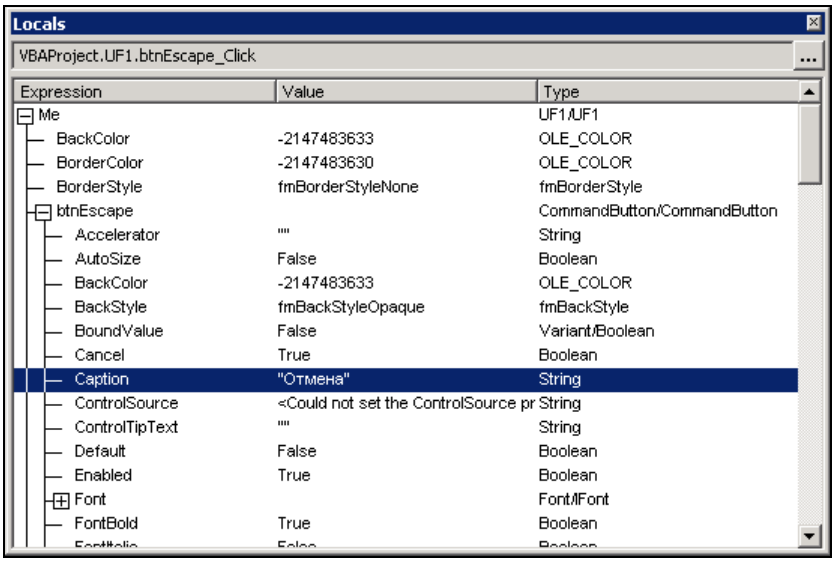


Рис. 6.4. Пример использования окна **Locals**

Чтобы поменять значение переменной или свойства, необходимо выделить нужную строку в окне **Locals**, а потом в этой строке аккуратно курсором выделить в столбце **Value** значение. Поверх старого значения можно впечатать новое. Если значение должно быть строковым, то при печати заключите его в кавычки (как в коде), а если это значение даты — то в символы решетки (#).

Через окно **Locals** можно также менять значения элементов массивов и коллекций.

6.2.6. Окно *Watches*

Окно **Watches** (его можно открыть так же, как и остальные, при помощи меню **View**) позволяет контролировать ход выполнения программы, производя "наблюдение" в соответствии с заданными вами условиями. При работе с ним

вам вначале потребуется определить значение, которое послужит сигналом к тому, чтобы вмешаться в ход выполнения программы. Это значение называется контролируемым выражением. Оно может быть совсем простым, например, `nResult = 10`, а может быть сложным, таким как:

```
InStr(oDoc.Fullname, "Document") <> 0
```

Создать контролируемое выражение можно так:

- щелкнуть по переменной, свойству или выражению в окне редактора кода правой кнопкой мыши и в контекстном меню выбрать **Add Watch**. Откроется диалоговое окно **Add Watch** для задания данного выражения;
- воспользоваться командой **Add Watch** в меню **Debug**;
- воспользоваться командой **Quick Watch** в меню **Debug**. В этом случае в окно **Watch** будет помещено автоматически сгенерированное выражение в зависимости от того, в каком месте кода находился у вас курсор ввода. В качестве условия для "срабатывания" в него будет помещено текущее значение переменной или свойства. Это контролируемое выражение в случае необходимости можно будет отредактировать;
- просто перетащить выражение из кода в окно **Watches**.

В любом случае откроется окно **Add Watch**. В нем вы должны написать (или дописать) контролируемое выражение, чтобы оно возвращало `True` или `False`, как и в конструкции `If...Else`, выбрать "область действия" данного контролируемого выражения (в виде процедуры или модуля), а также принять главное решение: что делать в ходе наблюдения. Вариантов у вас три:

- **Watch Expression** — ничего не делать (просто менять значение в столбце **Value** в окне **Watches**);
- **Break When Value Is True** — переводить программу в режим паузы, если контролируемое выражение "сработало" (его значение стало равно `True`);
- **Break When Value Changes** — переводить программу в режим паузы, если значение контролируемого выражения изменилось.

Окно **Watches** позволяет отследить происходящее в вашей программе даже в самых тяжелых случаях, когда понять, почему все работает именно так, а не иначе, очень сложно.

6.3. Перехват и обработка ошибок времени выполнения

Самое тяжелые для разработчика ошибки — это ошибки времени выполнения, которые могут возникнуть по самым разным причинам: пользователь

ввел недопустимое значение, файл с таким именем уже существует, сервер баз данных отказывается вставлять введенные пользователем значения, разорвано сетевое соединение и т. п. При возникновении ошибок времени выполнения обычно работа приложения аварийно завершается, а пользователю выдается встроенное сообщение, которое он вряд ли сможет расшифровать. Поэтому одна из самых трудоемких задач при создании программы на VBA — предусмотреть, какие ошибки могут возникнуть при работе пользователя и реализовать их обработку.

Общий принцип обработки ошибки выглядит так:

1. Перед опасным кодом (сохранение или открытие файла, возможность деления на ноль и т. п.) помещается команда:

   ```
   On Error GoTo метка_обработчика_ошибки
   ```

   например:

   ```
   Dim a As Integer, b As Integer, c As Integer
   On Error GoTo ErrorHandlerDivision
   c = a / b
   ```

2. Далее в коде программы помещается метка обработчика ошибки и программный код обработки:

   ```
   ErrorHandlerDivision:
       MsgBox "Ошибка при делении"
   ```

3. Поскольку в такой ситуации код обработчика ошибки будет выполняться даже в том случае, если ошибки не было, есть смысл поставить перед меткой обработчика команду `Exit Sub` (если это подпроцедура) или `Exit Function` (если это функция). Полный код нашей мини-программы может выглядеть так:

   ```
   Private Sub UserForm_Click()
       Dim a As Integer, b As Integer, c As Integer
       On Error GoTo ErrorHandlerDivision
       c = a / b
       Exit Sub
       ErrorHandlerDivision:
           MsgBox "Ошибка при делении"
   End Sub
   ```

   Как правило, если есть возможность исправить ошибку, то в обработчике ошибок ее исправляют (или предоставляют такую возможность пользователю), если нет — то выдают пользователю сообщение с объяснением и прекращают работу программы.

4. После выполнения кода обработчика ошибки вам нужно будет сделать выбор: либо продолжить выполнение той процедуры, в которой возникла ошибка, либо прекратить ее выполнение и передать управление вызвавшей ее процедуре. В вашем распоряжении три варианта:
   - еще раз выполнить оператор, вызвавший ошибку (если обработчик ошибки может определить характер возникший проблемы). Для этого достаточно в обработчик ошибок вставить команду `Resume`;
   - пропустить оператор, вызвавший ошибку. Для этой цели можно использовать команду `Resume Next`;
   - продолжить выполнение с определенного места в программе. Для этого используется команда `Resume` *`метка`*. Синтаксис работы с меткой — такой же, как в операторе `GoTo`.

Отметим еще несколько моментов, которые связаны с обработкой ошибок:

- чтобы вернуться в нормальный режим работы после прохождения опасного участка кода (отменить обработку ошибок), можно воспользоваться командой:

```
On Error GoTo 0
```

- в вашем распоряжении имеется также команда `On Error Resume Next`. Она предписывает компилятору просто игнорировать все возникающие ошибки и переходить к выполнению следующего оператора. На практике очень часто перед выполнением опасного оператора используется эта команда, а затем при помощи конструкции `Select Case` проверяется номер возникшей ошибки (через свойства объекта `Err`), и в зависимости от этого организуется дальнейшее выполнение программы.

Рассмотрим чуть подробнее специальный объект `Err`. У этого объекта есть два главных свойства и два метода.

- `Number` — это свойство показывает номер ошибки. Обычно оно и проверяется в обработчике ошибок, чтобы выяснить, какая именно ошибка возникла. Если номер ошибки равен 0, то ошибки не было.
- `Description` — текстовое описание возникшей ошибки. Именно оно по умолчанию возвращается пользователю (хотя пользователь вряд ли в нем что-либо поймет). Скорее это информация для разработчика.
- `Clear()` — этот метод очищает объект `Err` от старой информации об ошибках. То же самое делает и команда `On Error GoTo 0`.
- `Raise()` — позволяет сгенерировать ошибку в программе, передав ей номер и описание. Очень полезная возможность для проверки поведения программы, если смоделировать ситуацию с реальной ошибкой трудно.

Надо сказать, что обработка ошибок — это очень надежный, но и очень ресурсоемкий метод работы. Если в вашей программе есть возможность обойтись без генерации и перехвата ошибок (например, проверять вводимое пользователем значение при помощи встроенных функций), то лучше так и делать. В то же время наличие обработчиков ошибок, чтобы справляться с действительно аварийными ситуациями, — это большой плюс вашей программе (а зачастую и просто необходимость).

Задание для самостоятельной работы 6: Перехват ошибок времени выполнения

Подготовка:

1. Создайте новый файл Excel и сохраните его как C:\ErrorHandling.xls.
2. В ячейку A1 этого файла введите значение `"Результат деления:"`.
3. Щелкните правой кнопкой мыши по любой панели инструментов или меню и в открывшемся списке доступных панелей инструментов выберите **Элементы управления**.
4. На панели инструментов **Элементы управления** нажмите кнопку **Режим конструктора** (верхняя левая кнопка) и в этом режиме поместите на лист Excel новую кнопку. Для этого нужно щелкнуть по объекту **Кнопка** на панели инструментов **Элементы управления** и на листе определить местонахождение и размеры этой кнопки.
5. Щелкните по созданной вами кнопке правой кнопкой мыши и в контекстном меню выберите **Свойства**. Определите для нее свойства по вашему усмотрению. Выглядеть лист с кнопкой в итоге может, например, так, как показано на рис. 6.5.
6. В режиме конструктора щелкните по кнопке правой кнопкой мыши и в контекстном меню выберите **Исходный текст**. Откроется редактор кода Visual Basic с созданной процедурой для события `Click` данной кнопки. Поместите в него следующий код:

```
Private Sub CommandButton1_Click()
    Dim nNum1 As Integer
    Dim nNum2 As Integer
    Dim nResult As Integer
    nNum1 = InputBox("Введите первое число")
    nNum2 = InputBox("Введите второе число")
    nResult = nNum1 / nNum2
    Range("B1").Value = nResult
End Sub
```

7. Вернитесь на ваш лист Excel, выйдите из режима конструктора (щелкнув по кнопке **Выход из режима конструктора** на панели инструментов **Элементы управления**) и нажмите на созданную вами на листе кнопку. Убедитесь, что если вводить допустимые значения для делимого и делителя, то код работает правильно и выводит результат деления в ячейку B2.

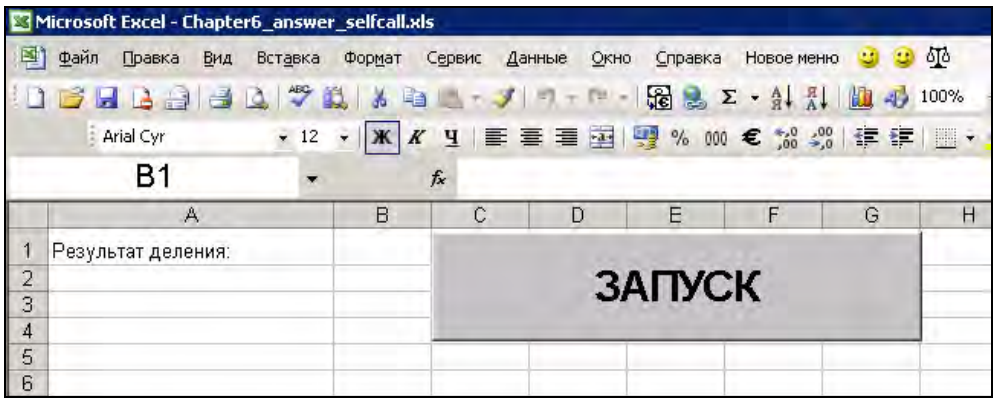


Рис. 6.5. Интерфейс вашей программы

ЗАДАНИЕ:

Измените эту программу таким образом, чтобы обеспечить защиту от ввода пользователем недопустимых значений (например, строковых значений делимого и делителя или значения делителя, равного 0).

Примечание

Не обязательно оставлять код по приему значений от пользователя и выполнению деления в обработчике события `Click` вашей кнопки. Этот код при желании можно перенести во внешние процедуры или функции.

Ответ к заданию 6

Существует множество вариантов решения этой задачи. Примеры приведены далее.

Так, например, может выглядеть решение с обработкой кода ошибки и повторным вызовом функцией самой себя:

```
Option Explicit
Private Sub CommandButton1_Click()
    Call subPrepare
End Sub
```

```
Public Sub subPrepare()
    Dim nReturnCode As Integer
    Dim nAnswer As Integer

    nReturnCode = fDiv()

    Select Case nReturnCode
    Case 1
        MsgBox ("Делить на ноль нельзя!")
        nAnswer = MsgBox("Повторить?", vbYesNo)
        If nAnswer = vbYes Then
            Call subPrepare
        Else
            Application.Quit
        End If
    Case 2
        MsgBox ("Нужно число!")
        nAnswer = MsgBox("Повторить?", vbYesNo)
        If nAnswer = vbYes Then
            Call subPrepare
        Else
            Application.Quit
        End If
    Case 3
        MsgBox ("Неизвестная ошибка")
        nAnswer = MsgBox("Повторить?", vbYesNo)
        If nAnswer = vbYes Then
            Call subPrepare
        Else
            Application.Quit
        End If
    End Select
End Sub

Function fDiv()
    On Error Resume Next
    Dim nNum1 As Integer
    Dim nNum2 As Integer
    Dim nResult As Integer

    nNum1 = InputBox("Введите первое число")
    nNum2 = InputBox("Введите второе число")

    nResult = CInt(nNum1) / CInt(nNum2)
```

```
    Select Case Err.Number
    Case 0
        Range("B1").Value = nResult
        fDiv = 0
    Case 11
        fDiv = 1
    Case 13
        fDiv = 2
    Case Else
        fDiv = 3
    End Select
End Function
```

А вот решение с обработкой кода ошибки и циклом:

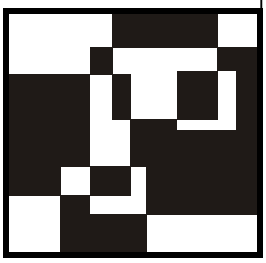
```
Private Sub CommandButton1_Click()
    Dim nNum1 As Variant
    Dim nNum2 As Variant
    Dim nResult As Integer
    Dim nError As Integer

    Do
        nNum1 = InputBox("Введите первое число:")
        On Error Resume Next
        nError = CInt(nNum1)
        If Err.Number = 13 Then
            MsgBox ("Нужно число")
            nNum1 = ""
        End If
        On Error GoTo 0
    Loop While (nNum1 = "")

    Do
        nNum2 = InputBox("Введите второе число:")
        On Error Resume Next
        nError = CInt(nNum2)
        If Err.Number = 13 Then
            MsgBox ("Нужно число")
            nNum2 = ""
        ElseIf nNum2 = 0 Then
            MsgBox ("Делить на ноль нельзя!")
            nNum2 = ""
        End If
        On Error GoTo 0
    Loop While (nNum2 = "")
```

```
    nResult = nNum1 / nNum2

    Range("B1").Value = nResult

End Sub
```

Еще один вариант решения вообще не допускает возникновения ошибок:

```
Private Sub CommandButton1_Click()
    Dim nNum1 As Variant
    Dim nNum2 As Variant
    Dim nResult As Integer

    Do
        nNum1 = InputBox("Введите первое число:")
        If IsNumeric(nNum1 & "") Then Exit Do
        MsgBox "Нужно число"
    Loop

    Do
        nNum2 = InputBox("Введите второе число:")
        If IsNumeric(nNum2 & "") Then
            If Int(nNum2) <> 0 Then Exit Do
            MsgBox "Делить на ноль нельзя!"
        Else
            MsgBox "Нужно число"
        End If
    Loop

    nResult = nNum1 / nNum2

    Range("B1").Value = nResult

End Sub
```

ГЛАВА 7



Работа с помощником

Для того чтобы вывести пользователю какую-либо информацию, в Office необязательно применять возможности форм с элементами управления или вызывать функцию `MsgBox()`. В приложения Office встроен помощник — анимированное изображение скрепки, щенка или котенка, которое пользователи чаще всего отключают. Однако применение его может быть очень удобным, особенно учитывая немодальность этого окна (т. е. пользователи могут продолжать работать в то время, когда помощник выводит им какие-то подсказки). Для работы с помощником используются два объекта — `Assistant` и `Balloon`. `Assistant` — это сам помощник, `Balloon` — это окно рядом с ним, в которое выводятся ваши сообщения (рис. 7.1).

Пример работы с объектом `Assistant` из программы на VBA может выглядеть так:

```
Assistant.On = True
Assistant.Visible = True
Assistant.Animation = msoAnimationThinking
```

Рис. 7.1. Вам помогает Бобик (английский псевдоним — Rocky)

Первая команда включает помощника (на случай, если он отключен), вторая — делает его видимым, третья — заставляет его "думать" (т. е. вызывает подходящий мультик). Далее приведен перечень наиболее важных свойств объекта `Assistant`.

- `Animation` — это свойство позволяет запустить "мультик" для нашего героя (думать, искать, отправлять почту, что-то делать и т. п.). По окончании мультика он останется в положении, определяемом этим свойством. Если какое-то действие для вашего героя не определено, ошибки не возникнет — оно будет просто проигнорировано.
- `FeatureTips` — это свойство переводит помощника в автоматический режим работы, заставляя его подсказывать вам более эффективные (по мнению разработчиков Office) пути выполнения различных операций.
- `FileName` — при помощи этого свойства вы можете выбрать помощника. По умолчанию выбирается щенок Rocky. Можно воспользоваться и другими помощниками — любым из семи, которые находятся в стандартной поставке Office, или сотнями других, которых можно найти в Интернете (но тогда их нужно будет предварительно скопировать в каталог к другим файлам помощников с расширением acs). Например, чтобы вызвать вместо щенка котенка, можно использовать такой код:

  ```
  Assistant.FileName = "Offcat.acs"
  ```

- `GuessHelp` — при включении этого свойства помощник начинает выводить список тех разделов справки, которые, по его мнению, могут пригодиться пользователю.
- `KeyboardShortcutTips` — помощник будет выводить в автоматический режим информацию о клавиатурных комбинациях, которые могут быть использованы для выполнения различных действий.
- `Left` — помощник "перепрыгнет" налево и будет располагаться в левой части экрана приложения.
- `MoveWhenInTheWay` — помощник автоматически будет уходить в сторону, если пользователь что-то делает на том месте, где находится помощник.
- `NewBalloon` — это свойство автоматически возвращает новый объект `Balloon` (т. е. окно "пузыря", в котором будет выводиться ваш текст для пользователя). Например, для создания "пузыря" можно использовать такой код:

  ```
  Dim Bal1
  Set Bal1 = Assistant.NewBalloon
  ```

  Чтобы `Balloon` был виден, необходимо вызвать его метод `Show()`:

  ```
  Bal1.Show
  ```

Собственно говоря, "пузырь" можно вообще не создавать, а воспользоваться конструкцией вида:

```
Assistant.NewBalloon.Heading = "У этого сотрудника фамилии не будет?"
```

- `On` — это свойство, как видно из нашего первого примера, включает помощника, если он отключен, или отключает, если в нем отпала необходимость.
- `SearchWhenProgramming` — определяет, будет ли помощник помогать во время работы редактора VBA.
- `Sounds` — включает или отключает звуковые эффекты.
- `TipOfDay` — помощник будет выдавать какой-нибудь совет из специального набора при каждом запуске приложения Office.
- `Top` — задает расстояние от верхнего края экрана до местонахождения помощника.
- `Visible` — позволяет показать или спрятать помощника.

При помощи методов помощника вы можете вызвать стандартное окно для поиска по справке (`Help()`), запустить мастер получения ответов (`StartWizard()`), переместить помощника в другое место (`Move()`) и т. п.

После того, как вы создали и настроили помощника, следующий этап — работа с объектом `Balloon`, т. е. с пузырем. Как он создается, мы уже видели. Теперь рассмотрим его основные свойства.

- `BalloonType` — позволяет настроить тип пузыря, в качестве значения может использоваться одна из трех констант. При помощи этого свойства определяется, будет ли пузырь содержать набор кнопок, нумерованный или маркированный список.
- `Button` — позволяет выбрать набор кнопок в нижней части пузыря.
- `Callback` — это свойство позволяет привязать к пузырю процедуру, которую пользователь сможет запустить из него.
- `Checkboxes` — позволяет разместить в пузыре переключатели.
- `Heading` — заголовок пузыря, в котором можно не только поместить текст, но и вставить изображение или выбрать цвет шрифта.
- `Icon` — свойство, задающее иконку в верхнем левом углу пузыря.
- `Labels` — позволяет привязать метки к кнопкам или элементам списков.
- `Mode` — задает режим работы баллона. В вашем распоряжении следующие значения:
  - `msoModeModal` (по умолчанию) — пользователю придется закрыть пузырь, прежде чем он сможет продолжить работу с приложением;

- `msoModeModeless` — пользователь может продолжить работу в приложении, не реагируя на пузырь;
- `msoModeAutoDown` — пользователь должен будет щелкнуть мышью по любому месту в приложении, чтобы пузырь пропал.

❒ `Text` — свойство, определяющее, что будет выводить пользователю пузырь. Помимо обычного текста, можно также выводить изображения и менять цвет шрифта.

Если при помощи свойства `Button` мы определили специальный набор кнопок, то, скорее всего, нам потребуется получить информацию о том, на какую кнопку нажал пользователь. Делается это очень просто:

```
intButton = Bal1.Show()
```

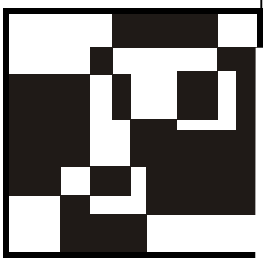
или напрямую:

```
Select Case Bal1.Show()
    ...
```

Второй метод объекта `Balloon` — метод `Close()`. Он просто закрывает наш пузырь с текстом.

Еще раз скажем, что ограничиваться только теми помощниками, которые идут в стандартной поставке Microsoft Office, совершенно необязательно. Вы можете установить на радость себе и окружающим множество альтернативных помощников, которые в большом количестве водятся в Интернете. Например, несколько сотен помощников можно бесплатно скачать с сайта **www.agentry.net**.

ГЛАВА 8



Работа с панелями инструментов и меню

Очень часто в приложении VBA вам потребуются свои наборы меню и панелей инструментов вместо стандартных, предусмотренных приложением. Работу с меню и панелями инструментов обеспечивает коллекция `CommandBars`, которая находится в объекте `Application` (об этом важном объекте мы будем говорить в следующих главах). Коллекция `CommandBars` содержит, как ясно из названия, набор объектов `CommandBar` (этот объект представляет как панели инструментов, так и меню), каждый из которых, в свою очередь, — коллекцию `CommandBarControls`, а эта коллекция представляет из себя хранилище элементов, из которых и состоит меню. Таких элементов может быть три:

- `CommandBarButton` — кнопка или элемент меню, который используется для выполнения программы или подпрограммы;
- `CommandBarComboBox` — сложный элемент меню или панели управления (это может быть поле ввода, раскрывающийся список, поле со списком);
- `CommandBarPopup` — меню или вложенное меню.

Пример создания собственной панели инструментов может выглядеть очень просто:

```
Dim CBar1 As CommandBar
Set CBar1 = CommandBars.Add("Документы", msoBarTop)
CBar1.Enabled = True
CBar1.Visible = True
```

У нас появилась новая панель инструментов **Документы** (рис. 8.1), которую можно, к примеру, убрать через меню **Настройка | Панели инструментов**, однако пока она совершенно бесполезна: в ней нет ни одной кнопки. Для того чтобы на панели инструментов была кнопка, необходимо добавить новый элемент типа `CommandBarControl` (одного из трех типов, перечисленных ранее) в коллекцию `CommandBarControls` для этого меню.

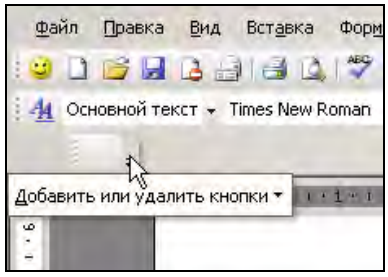


Рис. 8.1. Пустая панель инструментов **Документы**

Но вначале расскажем чуть подробнее о методе `Add()` коллекции `CommandBars`, при помощи которого мы создали новую панель инструментов. В нашем примере вызов этого метода выглядел так:

```
Set CBar1 = CommandBars.Add("Документы", msoBarTop)
```

Первый параметр (`"Документы"`) — имя объекта (название панели). Второй параметр (`msoBarTop`) — либо положение пристыкованной панели (`msoBarTop`, `msoBarBottom`, `msoBarLeft`, `msoBarRight`), либо знак того, что панель непристыкована (`msoBarFloating`), либо вообще указание на то, что это контекстное меню, которое до щелчка правой кнопкой мыши не видно (`msoBarPopup`). Свойство `Visible` для контекстного меню неприменимо.

Как будет выглядеть в итоге панель — как меню или как панель инструментов — зависит от того, какие элементы вы туда поместите.

Рассмотрим некоторые важные свойства и методы объекта `CommandBar`.

- `BuiltIn` — это свойство определяет, является ли данная панель/меню встроенной для этого приложения (т. е. предусмотренной в нем разработчиками приложения Office). Менять значение этого свойства нельзя. Его очень удобно использовать для того, чтобы убрать все стандартные меню или, наоборот, убрать все свои меню, оставив только стандартные.
- `Context` — определяет, где именно находится программный код для вашего меню (в Normal.dot, файле документа и т. п.). Можно использовать для проверок в случае потенциальной возможности вызова разных меню с одинаковыми именами.
- `Controls` — через это свойство можно получить коллекцию элементов управления `CommandBarControls`, которая нам, скорее всего, потребуется для работы с кнопками или элементами меню.
- `Enabled` — включение или отключение панели.
- `Height`, `Left`, `Top` и `Width` — очевидные свойства, относящиеся к расположению панели в окне приложения.

- `Index`, `Name` и `NameLocal` — эти свойства позволяют найти нужную нам панель в коллекции `CommandBars`. `Name` — это программное имя объекта, `NameLocal` — имя, которое будет видно пользователю, `Index` — номер данной панели в коллекции.

Примечание

Просмотреть номера всех встроенных панелей и меню в Office можно при помощи кода:

```
Dim cBar As CommandBar
For Each cBar In CommandBars
    Debug.Print cBar.Index & vbTab & cBar.Name
Next
```

Если вы хотите поместить новый элемент в стандартное меню Office, то номер стандартного меню в коллекции `CommandBars` — 41.

- `Protection` — позволяет запретить пользователю убирать старые кнопки из этой панели или размещать на ней новые.
- `Type` — наверное, самое важное свойство. Определяет, чем будет данная панель (панелью инструментов, обычным меню или контекстным меню, открывающимся по щелчку правой кнопкой мыши). Однако это свойство доступно только для чтения.
- `Visible` — это свойство определяет, будет ли панель инструментов видимой. Для вновь созданных панелей инструментов по умолчанию используется значение `False`, т. е. панель инструментов не видна.

После того как создание панели завершено, необходимо разместить на ней элементы управления. Например, создание кнопки на панели инструментов может выглядеть так:

```
Dim But1 As CommandBarControl
Set But1 = CBar1.Controls.Add(msoControlButton)
But1.Caption = "Кнопка 1"
```

Вроде бы ничего не произошло — перед нами та же пустая панель. Однако если навести указатель мыши на начало панели, можно увидеть пустую кнопку, которая называется **Кнопка 1** (рис. 8.2).

Как видно из кода, мы использовали для создания элемента управления (кнопки в панели инструментов) метод `Add()` коллекции `Controls` объекта `CommandBar` (у нас он называется `CBar1`). Свойства и методы у этой коллекции стандартные, как у множества других коллекций.

- `Application` — возвращает ссылку на объект приложения (Word, Excel и т. п.).

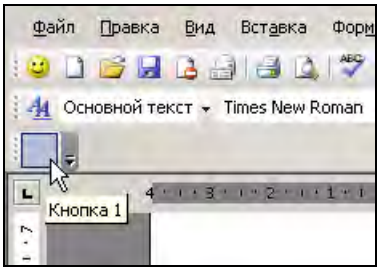


Рис. 8.2. На панели инструментов **Документы** появилась почти невидимая **Кнопка 1**

- `Count` — позволяет узнать, сколько всего элементов управления помещено в эту коллекцию.
- `Item` — позволяет по индексу (номеру) получить ссылку на объект элемента управления `CommandBarControl` в этой коллекции.
- `Add()` — позволяет добавить элемент управления в эту коллекцию (удаление элемента управления производится при помощи метода `Delete()` самого элемента управления).

Рассмотрим подробнее метод `Add()` коллекции `Controls`. Этот метод принимает пять необязательных параметров, из которых первые два параметра очень важны. Первый параметр `Type` определяет тип передаваемого в коллекцию элемента управления. Таких типов пять:

- `msoControlButton` — кнопка (т. е. в итоге получится панель инструментов);
- `msoControlEdit` — поле для ввода текста;
- `msoControlDropdown` — раскрывающийся список;
- `msoControlComboBox` — комбинированный список;
- `msoControlPopup` — пункт меню.

Например, чтобы вместо нашей кнопки был создан начальный пункт раскрывающегося меню (рис. 8.3), в нашем коде нужно изменить параметр метода `Add()`:

```
Dim But1 As CommandBarControl
Set But1 = CBar1.Controls.Add(msoControlPopup)
But1.Caption = "Меню 1"
```

Второй важный параметр метода `Add()` — параметр `Id`. Этот параметр позволяет привязать создаваемый элемент к уже имеющемуся в системе встроенному элементу управления. Например, чтобы добавить кнопку печати, этот параметр должен быть равен 4, а чтобы добавить кнопку предварительного просмотра документа, этот параметр должен быть равен 5. Если оставить этот

параметр пустым или указать для него значение 1, то будет создан пользовательский элемент управления, не привязанный ни к каким встроенным. К сожалению, значения этого параметра для встроенных элементов управления никак не документированы. Определить нужное значение можно или подбором, или просмотром значений свойства `Id` для имеющихся элементов управления (например, при помощи окна **Locals**).

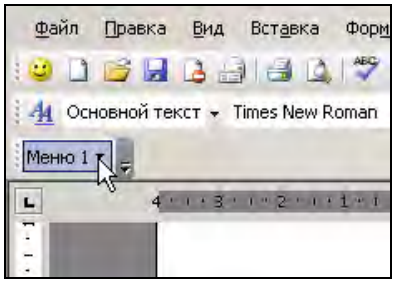


Рис. 8.3. Пункт меню на созданной нами панели инструментов

Остальные параметры этого метода относятся к созданию идентификатора элемента управления, его положению относительно других элементов управления и к тому, должен ли он быть постоянным или временным.

Конечно, работа с панелью управления/меню на этом не завершается. Нам потребуется еще донастроить свойства элементов управления. Например, для кнопки из первого примера нам, как минимум, нужно определить изображение (иконку), которое будет на кнопке, и процедуру, которая будет вызываться при нажатии на нее.

Важнейшие свойства и методы объекта `CommandBarButton` приводятся далее.

- `Caption` — надпись на элементе управления. Для кнопки выводится в виде всплывающей подсказки, а для пункта меню — название пункта.
- `Enabled` — определяет, включен или отключен данный элемент управления. Обычно используется для предупреждения ошибок пользователя.
- `FaceId` (только для кнопок) — позволяет использовать системную картинку для кнопки (не назначая ей соответствующей функции). Например, значение 4 присвоит кнопке изображение принтера. В Word и Excel встроено несколько тысяч иконок, и поэтому вместо создания новой иконки всегда есть возможность подобрать готовую. К сожалению, номера иконок тоже нигде не документированы, и вам придется выбирать подходящую методом перебора. Можно также создать для целей выбора иконки панели инструментов с несколькими сотнями кнопок, каждой из которых будет присвоено значение `FaceId`, увеличенное на 1 по отношению к предыдущей.

- `Id` — идентификатор встроенной функции, назначенной этой кнопке. Если вы назначили этой кнопке пользовательскую процедуру, то значение `Id` будет всегда равно 1.
- `Index` — номер элемента управления в коллекции `Controls`. Используется для выполнения служебных операций с элементами управления.
- `Mask` — позволяет наложить на рисунок объекта маску для показа только части рисунка. Маска выглядит как отдельный рисунок, прозрачные части которого должны быть белыми, а непрозрачные — черными.
- `OnAction` — самое важное свойство элемента управления. Его значение используется, когда элемент управления активизируется (щелчок по кнопке или пункту меню, завершение ввода текста в текстовом поле, выбор нового значения в списке). Предназначено для указания запускаемой процедуры или внешнего приложения (COM Add In). Наример:

  ```
  But1.OnAction = "MySub"
  ```

- `Parameter` — это свойство можно использовать для передачи параметров вызываемой подпроцедуре или просто для хранения своих данных. Работает с типом данных `String`.
- `Picture` — это свойство позволяет назначить рисунок (иконку) кнопке панели инструментов. Чаще используется не оно, а свойство `FaceId`. Если есть необходимость, то нужные иконки можно подобрать из большой коллекции, которая есть в Visual Studio, или воспользоваться одним из свободно доступных генераторов иконок.
- `ShortcutText` — текст с описанием клавиатурного сочетания, назначенного этой кнопке или пункту меню.
- `State` — вид кнопки (обычная, утопленная или обведенная рамкой).
- `Style` — еще одно свойство, влияющее на внешний вид. Позволяет определить, как будет выглядеть кнопка: будет показана только иконка, только надпись или и то, и другое вместе в разных вариантах.
- `ToolTip` — определяет текст всплывающей подсказки. По умолчанию "всплывает" значение свойства `Caption`.
- `Type` — возвращает тип элемента управления (только для чтения).
- `Visible` — определяет, будет этот элемент управления видимым или нет.
- `Delete()` — этот метод позволяет удалить кнопку из коллекции кнопок;
- `Execute()` — запускает на выполнение то, что определено при помощи свойства `OnAction`;
- `Reset()` — возвращает к исходным параметрам кнопки после внесенных изменений.

У объекта `CommandBarButton` есть единственное событие — `Click`. Оно также позволяет определить реакцию на нажатие кнопки, но работать с ним сложнее, чем со свойством `OnAction`.

В принципе, для работы с панелями инструментов этого вполне достаточно. Однако раскрывающиеся и контекстные меню устроены несколько сложнее. В раскрывающемся меню вам потребуется еще определить вложения элементов, а в контекстном меню — привязать это меню к какому-то объекту.

Для работы с вложенными меню используется точно та же коллекция `Controls`, которой мы уже пользовались. Единственное отличие в том, что эта коллекция `Controls` принадлежит не объекту `CommandBar`, а объекту `CommandBarPopup`, т. е. другому меню. В нашем примере вложение будет выглядеть так:

```
'Создаем стандартный объект CommandBar
Dim CBar1 As CommandBar
Set CBar1 = CommandBars.Add("Документы", msoBarTop)
CBar1.Enabled = True
CBar1.Visible = True

Dim Menu1 As CommandBarPopup
Dim SubMenu1 As CommandBarPopup
Dim SubMenu1Item As CommandBarButton

'Создаем верхнее меню
Set Menu1 = CBar1.Controls.Add(msoControlPopup)
Menu1.Caption = "Меню 1"

'Создаем вложенное меню
Set SubMenu1 = Menu1.Controls.Add(msoControlPopup)
SubMenu1.Caption = "Подменю 1"

'Создаем элемент во вложенном подменю и назначаем ему процедуру Proc1
Set SubMenu1Item = SubMenu1.Controls.Add(msoControlButton)
SubMenu1Item.FaceId = 5
SubMenu1Item.Caption = "Элемент подменю"
SubMenu1Item.OnAction = "Proc1"
```

У вас должно получиться то, что изображено на рис. 8.4.

Конечно, можно добавлять элементы не только в свои меню, но и во встроенные. Добавление происходит точно так же, а найти нужное встроенное меню можно при помощи цикла `For Each` и проверки значения свойства `Name`.

Контекстные меню (в справке VBA *shortcut menus*) — это меню, которые открываются по щелчку правой кнопкой мыши. Работа с ними выглядит так:

```
Set CBar1 = CommandBars.Add("Мое контекстное меню", msoBarPopup, , _
            True)
Set MenuItem1 = CBar1.Controls.Add
MenuItem1.FaceId = 3
MenuItem1.Caption = "Элемент меню 1"
Set MenuItem2 = CBar1.Controls.Add
MenuItem2.FaceId = 5
MenuItem2.Caption = "Элемент меню 2"
```

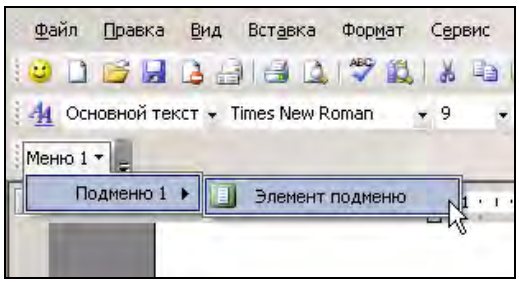


Рис. 8.4. Пример вложенного меню

Как мы видим, все очень просто и стандартно. Однако если выполнить этот код, то никакого контекстного меню не появится: необходимо еще добавить вызов метода `ShowPopup()`:

```
CBar1.ShowPopup
```

Тогда контекстное меню возникнет в том месте, где сейчас находится указатель мыши (можно передать этому методу координаты места появления). Конечно, этот метод нужно вызывать из обработчика события, связанного с правой кнопкой мыши, а его как раз и нет для многих объектов. Например, в Excel для листа есть событие `BeforeRightClick`, а для документа Word такого события нет. Но в этом случае у нас останется возможность добавить свои пункты в стандартное контекстное меню.

Задание для самостоятельной работы 8: Работа с панелями инструментов, меню и помощником

ЗАДАНИЕ:

1. Создайте и сделайте видимой в Word новую панель инструментов с двумя кнопками: **Показать помощника** и **Отключить помощника**. Она может выглядеть, например, так, как представлено на рис. 8.5.

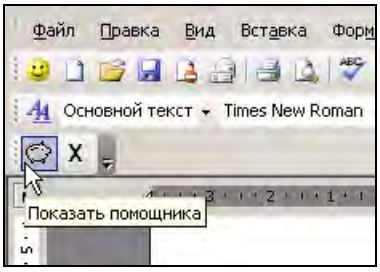


Рис. 8.5. Новая панель инструментов Word

При нажатии на кнопку **Показать помощника** должен появляться помощник, а при нажатии на кнопку **Отключить помощника** он должен отключаться.

2. Сделайте так, чтобы при нажатии на кнопку **Показать помощника** появлялось дополнительное меню, аналогичное представленному на рис. 8.6.

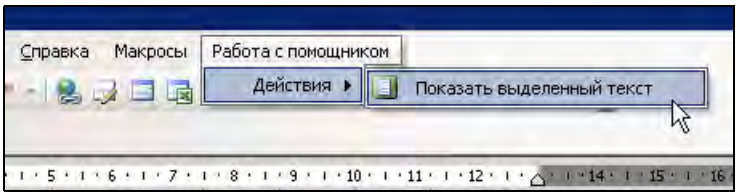


Рис. 8.6. Созданное программным образом новое меню

При выборе пункта меню **Показать выделенный текст** в окне помощника должен отображаться текст, который выделен в документе (рис. 8.7)

Рис. 8.7. Вызванный макросом помощник в действии

Ответ к заданию 8

Набор процедур, которые бы выполняли поставленные условия, может выглядеть так, как представлено далее (для создания панели инструментов нуж-

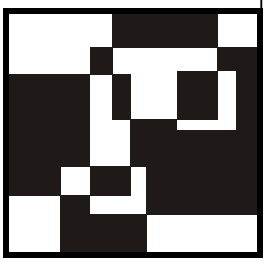
но запустить процедуру `ShowNewToolbar()`). Если возможности этих процедур нужны во всех документах Word, то лучше их поместить в модуль проекта Normal.dot.

```
Public Sub ShowNewToolbar()
'Исходная процедура, которая создает новую панель инструментов Assistant

    'Проверка — если панель с таким именем уже есть, создавать ее не надо
    Dim cBar As CommandBar
    For Each cBar In CommandBars
        If cBar.Name = "Assistant" Then
            cBar.Visible = True
            Exit Sub
        End If
    Next

    'Создаем панель управления
    Dim CBar1 As CommandBar
    Set CBar1 = CommandBars.Add("Assistant", msoBarTop)
    CBar1.Enabled = True
    CBar1.Visible = True

    'Помещаем на нее первую кнопку
    Dim But1 As CommandBarControl
    Set But1 = CBar1.Controls.Add(msoControlButton)
    But1.Caption = "Показать помощника"
    But1.FaceId = 52
    But1.OnAction = "ShowAssistant"

    'Помещаем на нее вторую кнопку
    Dim But2 As CommandBarControl
    Set But2 = CBar1.Controls.Add(msoControlButton)
    But2.Caption = "Отключить помощника"
    But2.FaceId = 103
    But2.OnAction = "DisableAssistant"
End Sub

Public Sub ShowAssistant()
'Процедура, которая запускается по нажатию кнопки "Показать помощника"

    'Включение помощника
    Assistant.On = True
    Assistant.Visible = True

    'Начинается код для создания нового меню

    'Проверяем, есть ли уже такой пункт в стандартном меню
    Dim cBarCont As CommandBarControl
```

```
    For Each cBarCont In CommandBars(41).Controls
        If cBarCont.Caption = "Работа с помощником" Then
            Exit Sub
        End If
    Next

    'Создаем меню
    Dim CBar1 As CommandBar
    Set CBar1 = CommandBars(41)
    CBar1.Enabled = True
    CBar1.Visible = True

    Dim Menu1 As CommandBarPopup
    Dim SubMenu1 As CommandBarPopup
    Dim SubMenu1Item As CommandBarButton

    'Создаем верхнее меню
    Set Menu1 = CBar1.Controls.Add(msoControlPopup)
    Menu1.Caption = "Работа с помощником"

    'Создаем вложенное меню
    Set SubMenu1 = Menu1.Controls.Add(msoControlPopup)
    SubMenu1.Caption = "Действия"

    'Создаем элемент во вложенном подменю и назначаем ему
    'процедуру TextToAssistant
    Set SubMenu1Item = SubMenu1.Controls.Add(msoControlButton)
    SubMenu1Item.FaceId = 5
    SubMenu1Item.Caption = "Показать выделенный текст"
    SubMenu1Item.OnAction = "TextToAssistant"
End Sub

Public Sub DisableAssistant()
'Процедура, которая отключает помощника и созданное меню

    'Этот код можно использовать для кнопки "Отключить помощника"
    Assistant.On = False

    Dim cBarCont As CommandBarControl
    For Each cBarCont In CommandBars(41).Controls
        If cBarCont.Caption = "Работа с помощником" Then
           cBarCont.Delete
            Exit Sub
        End If
    Next
End Sub
```

```
Public Sub TextToAssistant()
'Процедура, которая выводит в помощнике выделенный в документе текст

    Dim oBal1 As Balloon
    Set oBal1 = Assistant.NewBalloon
    oBal1.Text = Selection.Text
    oBal1.Show
End Sub
```

ГЛАВА 9



Работа с базами данных и применение объектной модели ADO

9.1. Зачем нужно работать с базами данных

Потребность в автоматизации (например, в создании программ VBA) возникает тогда, когда данных много. А если данных много, то они, скорее всего, будут храниться в базе данных — просто потому, что более удобного способа пока не придумано. Это относится к любым данным (в том числе к документам, графическим данным, архивам и т. п.).

Приложения Office и созданные на их основе программы при работе с данными могут быть очень полезными и сами по себе, но эффективность работы увеличивается многократно при сопряжении их с базами данных. Чаще всего в реальных приложениях Word используется для генерации отчетов на основе информации из базы данных, Excel — для анализа данных из базы, а Access — это сама по себе система управления базами данных (которая очень часто используется для построения клиентского интерфейса для внесения информации в клиент-серверные базы данных, такие как SQL Server и Oracle).

Потребности в обращении из приложений Office к базам данных возникают практически на любом предприятии. Очень часто приложение, которое изначально предназначалось для работы с данными в самом приложении (листе Excel, таблицы Word), по мере увеличения объема данных приходится переделывать под работу с клиент-серверными источниками. Поэтому в этой главе рассказывается о том, как можно подключаться к базам данных, скачивать и отображать в программе информацию из базы данных, вставлять новые записи, изменять или удалять существующие. Если у вас нет никакого опыта работы с базами данных, не пугайтесь. Как показывает опыт многих учебных

групп, знаний на уровне опытного администратора баз данных совсем не требуется. Освоить основные приемы работы с базами данных за несколько дней вполне способен каждый.

В этой главе речь пойдет об универсальных приемах работы с базами данных. Освоив их, вы сможете работать из приложений Office (и не только из них) с любыми базами данных: клиент-серверными, такими как Microsoft SQL Server, Oracle, IBM DB2, настольными, такими как Access, FoxPro, DBase, Paradox, и даже источниками данных, которые сами по себе базами не являются (например, иногда очень удобно подключиться к файлу Excel как к базе данных).

9.2. Что такое ADO

ADO расшифровывается как *ActiveX Data Objects* — набор программных объектов, построенных по технологии ActiveX (COM), которые позволяют получать данные из самых разных источников и управлять ими. Другие наборы программных объектов для доступа к источникам данных, которые часто используются в приложениях Office — это DAO и RDO, но эти программные объекты устарели и к использованию в современных приложениях не рекомендуются. В настоящее время появилась новая версия ADO — ADO.NET, которая сильно отличается от обычной ADO и предназначена для работы в .NET Framework. Однако по причине того, что ADO.NET:

- обязательно требует установленной .NET Framework (чего на многих старых компьютерах нет);
- обычными средствами с ADO.NET из редактора Visual Basic работать нельзя, требуется Visual Studio;
- отличается повышенной ресурсоемкостью.

ADO.NET в этой книге рассматриваться не будет.

ADO умеет работать с самыми разными драйверами для подключения к базам данных, например, с драйверами OLE DB и ODBC. Поскольку ADO построен по технологии COM, его можно использовать в любых COM-совместимых языках программирования (VC++, Visual Basic, Delphi, VBA, VBScript, JScript, ActivePerl и т. п.).

Сами программные объекты поставляются в наборе драйверов для подключения к базам данных, которые называются MDAC (Microsoft Data Access Components, компоненты доступа к данным Microsoft). Этот набор есть на любом компьютере под управлением Windows 2000, XP, 2003 (обычно сразу несколько версий). Самую свежую версию MDAC можно бесплатно скачать с Web-сайта фирмы Microsoft (**www.microsoft.com/download**). Настоятельно

рекомендуется отслеживать появление новых версий MDAC и устанавливать их на компьютерах пользователей.

К справке по ADO проще всего обращаться из Microsoft Access. Для этого нужно открыть Microsoft Access, создать в нем новую пустую базу данных, переключиться в окно редактора кода (при помощи клавиш <Alt>+<F11>) и нажать клавишу <F1>. Вторая строка в оглавлении справки — это справка по ADO. В нее входят **ADO Programmer's Guide** (рассказ о том, как можно выполнять различные действия), и **ADO Programmer's Reference** (справка по объектам ADO, их свойствам, методам и событиям) (рис. 9.1).

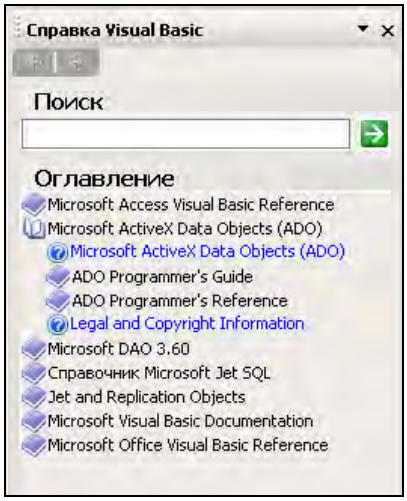


Рис. 9.1. Справка по ADO в Microsoft Access

Сама по себе объектная модель ADO очень проста и понятна. В нее включены всего три главных объекта.

- `Connection` — позволяет установить соединение с источником данных и управлять им. Все ошибки, которые возникают в ходе работы соединения, помещаются в сопутствующую коллекцию `Errors`.
- `Command` — представляет собой команду, при помощи которой производится выполнение определенной операции на источнике данных (выполнение запроса, хранимой процедуры, создание или изменение объекта, изменение данных и т. п.). Если источник данных SQL-совместимый, то объект `Command`, скорее всего, будет представлять команду SQL. Объекту `Command` сопутствует коллекция `Parameters` — параметры, которые передаются запросу или хранимой процедуре.
- `Recordset` — является набором записей, полученных с источника или сгенерированных другим способом. Этому объекту сопутствует коллекция

Fields, представляющая информацию о столбцах в этом наборе записей (имя, тип, размерность данных и т. п.), а также сами данные.

Для каждого из этих трех объектов предусмотрена также коллекция Properties, которая определяет свойства соединения, команды или набора записей соответственно.

Все объекты явно создавать необязательно. Так, например, при создании объекта Recordset можно в автоматическом режиме создать объект Connection.

Прежде чем приступить к работе с объектами ADO, необходимо добавить в ваш проект ссылку на необходимую библиотеку. Для этого в меню **Tools** выберите пункт **References** и установите флажок напротив строки **Microsoft ActiveX Data Objects** с нужным номером (в зависимости от того, какая версия библиотеки установлена на клиентских компьютерах, на которых будет работать ваше приложение). На момент создания этого курса последней версией MDAC была версия 2.8. Однако на практике рекомендуется выбирать более старую версию, например, версию 2.1, которая поставляется со всеми версиями Windows, начиная с Windows 2000. Функциональность разных версий практически одинакова, но при использовании этой версии вы сможете переносить свою программу на VBA с одного компьютера на другой, не беспокоясь о том, что на него нужно что-то доустанавливать.

9.3. Объект *Connection* и коллекция *Errors*

Создание объекта Connection выполняется очень просто. Например, чтобы подключиться к базе данных Northwind на сервере SQL Server с именем LONDON, можно использовать код типа:

```
Dim cn As New ADODB.Connection
cn.ConnectionString = "Provider=SQLOLEDB.1;Integrated
Security=SSPI;Initial Catalog=Northwind;Data Source=LONDON"
cn.Open
```

В принципе, этого вполне достаточно, чтобы подключиться к базе данных и создать объект соединения, который можно будет потом использовать. Однако у большинства пользователей возникает вопрос: а что написано в свойстве ConnectionString и как значение этого свойства можно написать самому?

Самый простой вариант — вообще ничего самому не писать. Значение для этого свойства можно сгенерировать в автоматическом режиме. Выглядит это очень просто:

1. Создаем любой пустой файл (например, текстовый). Для этого нужно щелкнуть правой кнопкой мыши на пустом месте в окне проводника Windows и в контекстном меню выбрать **New | Text Document** — рис. 9.2. И затем задать создавшемуся файлу любое название.

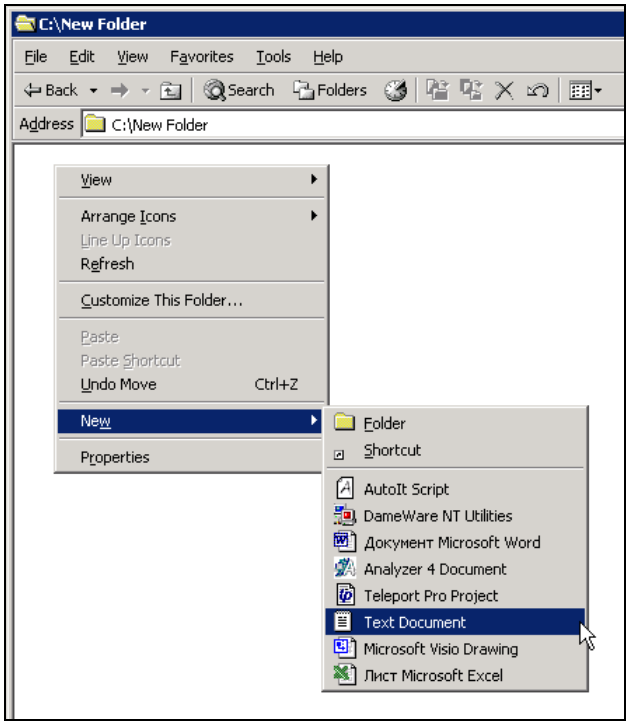


Рис. 9.2. Создание пустого текстового файла в Windows Explorer

2. Переименовываем файл так, чтобы у него было расширение udl (от *User Data Link* — пользовательское подключение к данным). После переименования убедитесь, что иконка для него изменилась (рис. 9.3).

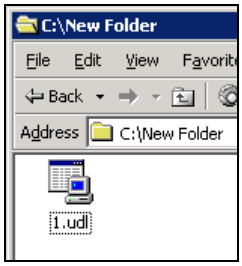


Рис. 9.3. Созданный файл 1.udl

Если она осталась такой же, как была для текстового документа, это значит, что реальное расширение для этого файла — txt, а не udl. В этом случае нужно в окне Windows Explorer в меню **Tools** выбрать **Folder Options**, в окне **Folder Options** перейти на вкладку **View** и снять флажок **Hide file**

extensions for known file types (скрывать расширения для известных типов файлов). После того, как вы выполните эту операцию, измените расширение файла на udl.

3. После того, как UDL-файл будет создан, просто щелкните по нему два раза левой кнопкой мыши. Откроется окно **Data Link Properties** с четырьмя вкладками.
4. На первой вкладке **Provider** выберите нужный тип базы данных (например, для базы данных SQL Server выберите **Microsoft OLE DB Provider for SQL Server**, для подключения к базе Oracle — **Microsoft OLE DB Provider for Oracle**, для подключения к базе Access — **Microsoft JET 4.0 OLE DB Provider**). Про подключение к листу Excel как к базе данных мы поговорим отдельно в *разд. 9.4*.
5. Далее перейдите на вкладку **Connection**. Для каждого типа базы данных эта вкладка выглядит по-своему. Например, для SQL Server она выглядит так, как представлено на рис. 9.4, а для Access — на рис. 9.5.

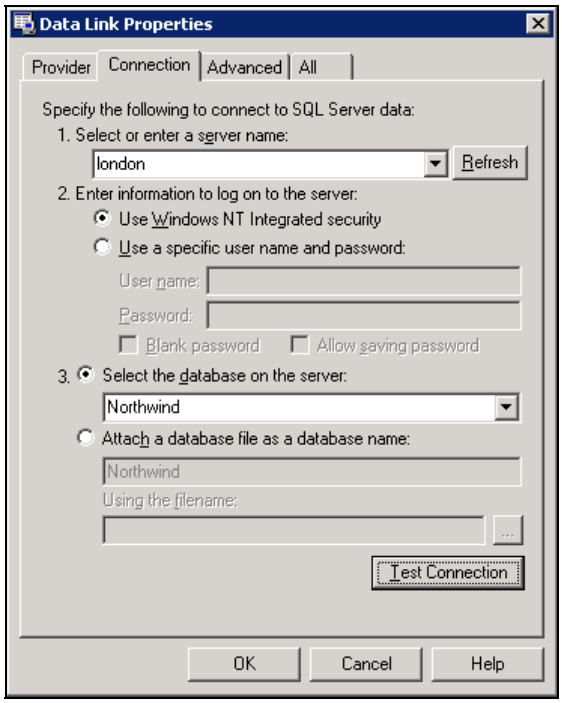


Рис. 9.4. Окно свойств соединения для подключения к SQL Server

Если вы не знаете, какие параметры вам нужно вводить в этом окне, спросите у вашего администратора базы данных. После того, как все парамет-

ры введены, рекомендуется нажать на кнопку **Test Connection**, чтобы проверить возможность подключения к базе данных. Если возникла ошибка, то нужно исправить введенные вами параметры свойств соединения. После этого нужно нажать кнопку **OK**, чтобы закрыть окно свойств соединения.

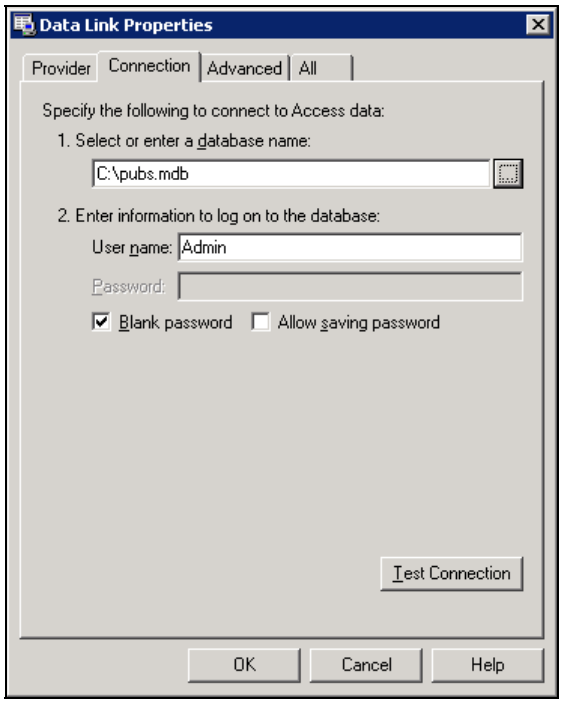


Рис. 9.5. Окно свойств соединения для подключения к базе данных Access

6. Последнее действие, которое нам потребуется сделать — щелкнуть правой кнопкой мыши по созданному UDL-файлу, в контекстном меню выбрать **Open With | Choose Program** и в появившемся списке выбрать **Notepad** (Блокнот) и нажать кнопку **OK**. Созданный вами файл откроется в блокноте. В меню **Format** в блокноте снимите флажок **Word wrap** и скопируйте в буфер обмена последнюю строку этого файла (рис. 9.6).

Внимание!

Если этой строке у вас есть русские буквы, то рекомендуется производить копирование при включенной русской раскладке клавиатуры, иначе русские символы будут заменены на знаки вопроса.

7. Осталось вставить скопированное значение в окно редактора кода как значение свойства `ConnectionString` и взять его в кавычки.

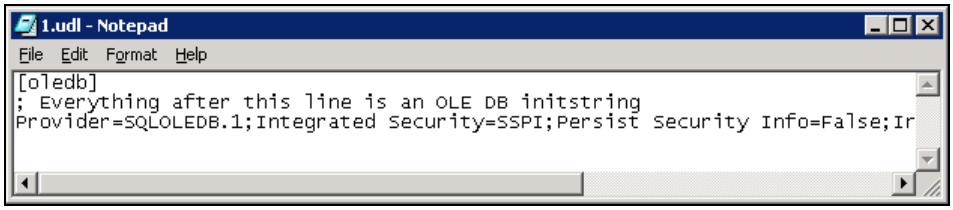


Рис. 9.6. Сгенерированная строка подключения

Конечно, вы вполне можете написать строку подключения *(connection string)* и вручную. Это просто набор параметров вида "*свойство=значение*", разделенных точкой с запятой (для значений в строке подключения кавычки не используются). Каждый из параметров означает следующее.

- `Provider` — драйвер для подключения к источнику данных. Для каждого типа источника данных (SQL Server, Access, Oracle) он свой. Мы с вами использовали драйвер OLE DB. Альтернатива ему — драйверы ODBC. Они работают медленнее и в основном используются для обеспечения обратной совместимости, но иногда без них не обойтись (например, когда подходящего драйвера OLE DB просто нет). Мы будем использовать подключение по ODBC к таблице на листе Excel.

  В принципе, в строке подключения значение для `Provider` можно не указывать вообще. Но тогда придется определить его как значение отдельного свойства объекта `Connection`.

- `Integrated Security` — в данном случае это свойство используется, поскольку мы применяем для подключения к SQL Server аутентификацию Windows. Если бы мы использовали аутентификацию SQL Server, то это свойство нам бы было не нужно, но вместо него нам потребовалось бы использовать два других свойства: `User ID` — идентификатор пользователя и `Password` — пароль. Если вы далеки от мира баз данных, просто узнайте необходимые значения у вашего администратора.

- `Data Source` — имя источника данных. В нашем случае это имя компьютера, на котором работает SQL Server. В других случаях оно могло бы быть именем экземпляра Oracle или файла базы данных Microsoft Access, все зависит от того, к какой базе данных вы подключаетесь.

- `Initial Catalog` — имя базы данных на этом сервере. В нашем случае это `Northwind`.

Когда вы генерируете строку подключения автоматически при помощи UDL-файла, в первый раз это может показаться долгим процессом. На самом деле это занимает не более минуты. Кроме того, при этом вы гарантируете, что в строке подключения не будет ошибок, и у вас есть возможность проверить

подключение к базе данных прямо из свойств UDL-файла (кнопка **Test Connection**).

Фактически все, что нужно для открытия соединения с базой данных, мы сделали: создали объект `Connection`, настроили для него свойство `ConnectionString` и вызвали метод `Open()`. Однако для справки приведем информацию об основных свойствах и методах этого объекта.

- `Provider` — это свойство позволяет определить драйвер, который будет использован для подключения к базе данных. Мы с вами определили такой драйвер внутри значения `ConnectionString`, но можно для этой цели использовать и отдельное свойство. Значения свойств `Provider` для подключения к разным источникам данных могут выглядеть так:
  - `"Microsoft.Jet.OLEDB.4.0"` — для подключений к файлам Access;
  - `"SQLOLEDB.1"` — для подключений к SQL Server (как в примере);
  - `"MSDAORA.1"` — для подключений к серверу Oracle;
  - `"ADsDSOObject"` — для подключения к базе данных службы каталогов Windows.
- `ConnectionString` — это главное свойство объекта `Connection`. Оно определяет параметры подключения к источнику. Как именно работать с ним, мы уже рассмотрели.
- `Open()` — этот метод позволяет открыть соединение с базой данных. Строку подключения можно не настраивать отдельно как свойство объекта `Connection`, а просто передавать ее этому методу как параметр.
- `Close()` — позволяет закрыть соединение. Учтите, что объект соединения при этом из памяти не удаляется. Чтобы полностью избавиться от этого объекта, можно использовать код:

  ```
  cn.Close
  Set cn = Nothing
  ```

  или просто:

  ```
  Set cn = Nothing
  ```

  а разрыв соединения произойдет автоматически.

Для объекта `Connection` предусмотрено множество других свойств и методов, однако здесь они рассматриваться не будут (за дополнительной информацией можно обратиться к документации или к учебным курсам Microsoft). Единственное свойство, которое обязательно нужно знать, — это свойство `Errors`, которое возвращает коллекцию объектов `Error` — ошибок. Ошибки при установке или работе соединения встречаются очень часто (неверно введен пароль или имя пользователя, у пользователя недостаточно прав на подключе-

ние, невозможно обратиться к компьютеру по сети и т. п.), поэтому настоятельно рекомендуется реализовывать в программе обработку ошибок. Самый простой вариант реализации обработчика ошибок может выглядеть так:

```
Dim cn As ADODB.Connection
Set cn = CreateObject("ADODB.Connection")
cn.Provider = "SQLOLEDB"
cn.ConnectionString = "User ID=SA;Password=password;" _
    & "Data Source = LONDON1;Initial Catalog = Northwind"
On Error GoTo CnErrorHandler
cn.Open
Exit Sub
CnErrorHandler:
    For Each ADOErr In cn.Errors
        Debug.Print ADOErr.Number
        Debug.Print ADOErr.Description
    Next
```

На практике при возникновении ошибки пользователю предлагается ее исправить и еще раз произвести подключение.

Для получения информации о том, почему возникла ошибка, используется специальный объект `ADOError` (при возникновении ошибки он создается автоматически). Далее представлены самые важные свойства этого объекта.

- `Description` — описание ошибки. Обычно наиболее важная информация содержится именно в описании.
- `Number` — номер ошибки. По номеру удобно производить поиск в базе знаний Microsoft (**www.microsoft.com/support**) и в Интернете.
- `Source` — источник ошибки. Эта информация полезна только в том случае, если в коллекции `Errors` могут оказаться ошибки из разных источников.
- `SQLState` и `NativeError` — информация о возникшей ошибке, которая пришла с SQL-совместимого источника данных.

9.4. Подключение к таблице на листе Excel

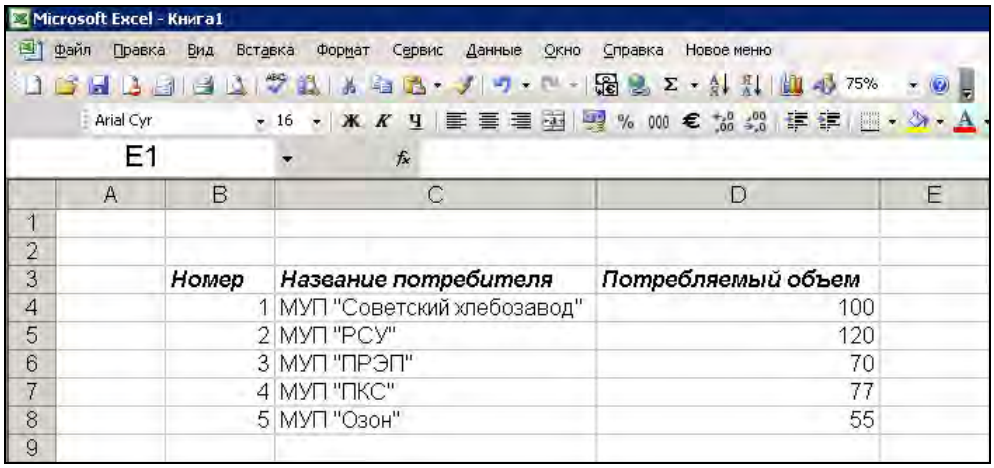
Очень часто в практической работе возникает необходимость подключиться к таблице на листе Excel как к базе данных. Конечно, можно работать и с объектной моделью Excel *(см. гл. 11)*, но использование объектов ADO дает значительные преимущества:

- намного проще и удобнее производить поиск записи, вставку новых записей в таблицу, изменение существующих записей. Объекты ADO изначально проектировались именно для этих целей;

- объектную модель Excel можно использовать только в Excel, а объекты ADO универсальны и могут применяться для подключения к любым источникам данных. Если вы используете объекты ADO, то можете использовать фактически одно и то же приложение как для работы с данными в Excel, так и для работы с информацией в "большой" базе данных, например, в SQL Server или Oracle. Ситуация, когда часть информации находится в базе данных, а другая часть — в книге Excel, встречается на практике очень часто.

Подключиться к таблице на листе Excel совсем не сложно, но самостоятельно догадаться до всей последовательности действий бывает трудно. Поэтому далее приведена пошаговая последовательность действий.

Предположим, что у нас есть книга Excel, которая называется Fact.xls и лежит в корневом каталоге диска C:. На первом листе этой книге есть несложная таблица, представленная на рис. 9.7.



	A	B	C	D	E
1					
2					
3		*Номер*	*Название потребителя*	*Потребляемый объем*	
4		1	МУП "Советский хлебозавод"	100	
5		2	МУП "РСУ"	120	
6		3	МУП "ПРЭП"	70	
7		4	МУП "ПКС"	77	
8		5	МУП "Озон"	55	
9					

Рис. 9.7. Таблица в Excel, к которой нужно обратиться средствами ADO

Нам необходимо подключиться к этой таблице, как к базе данных.

Первый этап — это подготовка. Иногда можно обойтись и без нее (если лист Excel — это одна таблица). На практике же часто бывает так, что на листе у нас несколько таблиц, или таблица с комментариями, или под таблицей посчитаны итоги и т. п. Чтобы не смущать Excel, лучше явно указать нашу таблицу. Сделать это очень просто: нужно ее выделить (в нашем случае — диапазон с B3 по D8) и присвоить выделенному диапазону имя. Для этого в Excel в меню **Вставка** нужно выбрать **Имя | Присвоить** и ввести имя. В нашем случае мы присвоили имя **Volumes** (рис. 9.8).

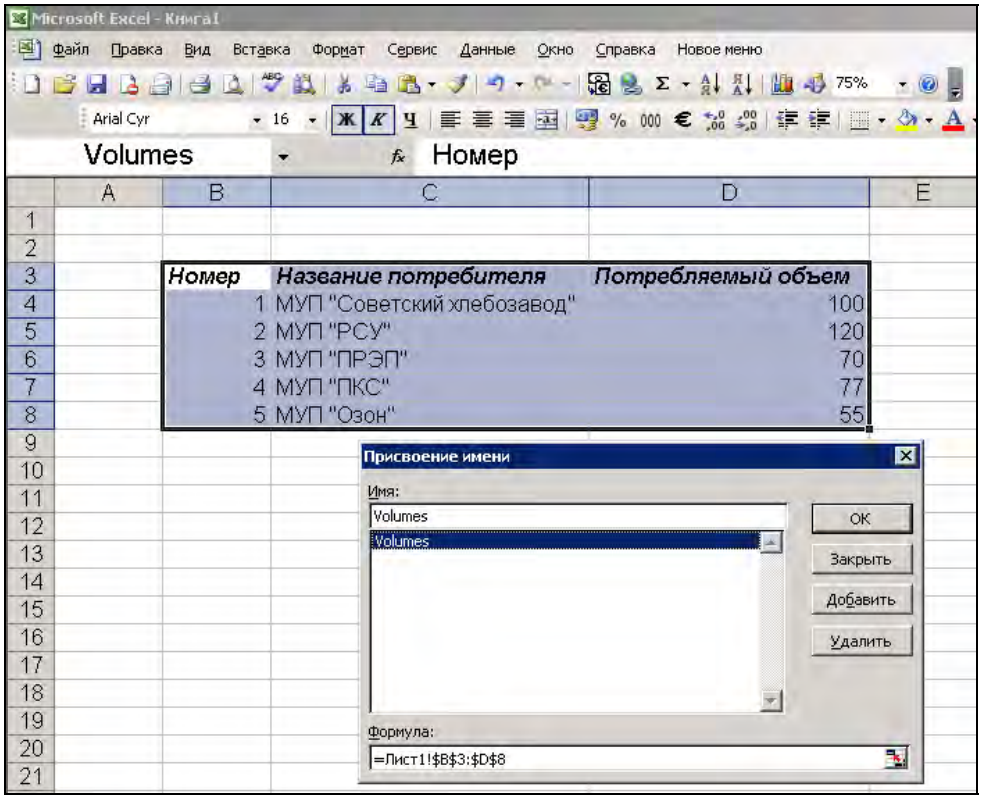


Рис. 9.8. Присвоение имени диапазону

Обратите внимание, что диапазон нужно выбирать вместе с названиями столбцов.

После того, как имя присвоено, документ Excel нужно сохранить и можно закрывать — он больше нам не нужен.

Далее по плану нужно было бы создать UDL-файл и настроить в нем подключение к нашему файлу C:\Fact.xls. Однако напрямую из UDL-файла можно работать только с драйверами OLE DB, а нужного нам драйвера, к сожалению, нет (Microsoft JET 4.0 OLE DB Provider хочет работать только с файлами mdb). Поэтому делаем еще один подготовительный шаг — создаем источник данных ODBC (поскольку драйвер ODBC для подключения к Excel есть). Первое действие — в **Панели управления** Windows открываем **Administrative Tools** (Средства администрирования) и два раза щелкаем по иконке **Data Sources (ODBC)** (Источники данных ODBC). Откроется окно, представленное на рис. 9.9.

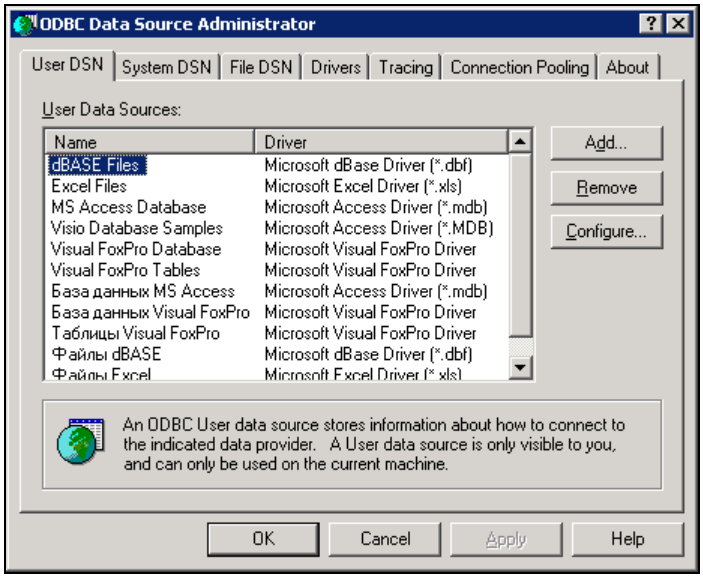


Рис. 9.9. Окно управления источниками данных ODBC

В вашем распоряжении три вкладки с DSN (*Data Source Name* — источников данных ODBC):

- **User DSN** (пользовательские источники данных) — информация об этих источниках данных хранится в части реестра, специфической для пользователя, поэтому эти источники данных доступны только тому пользователю, который их создал;
- **System DSN** (системные источники данных) — информация об этих источниках данных хранится в общей части реестра и доступна для всех пользователей на этом компьютере;
- **File DSN** (файловые источники данных) — информация об этих источниках данных записывается в файл в файловой системе.

Чаще всего используются системные источники данных, поэтому переходим на вкладку **System DSN** и нажимаем кнопку **Add**.

На первом экране мастера создания нового подключения нам потребуется ввести информацию о типе драйвера, который мы хотим использовать. Выбираем, конечно, **Microsoft Excel Driver** и нажимаем кнопку **Finish**. Но создание источника данных на этом далеко не кончилось.

На следующем экране мастера нам потребуется:

- в поле **Data Source Name** ввести имя источника данных. Можно ввести любое имя — главное, чтобы вы его не забыли. Введите имя **ExcelVolumes**;

- нажать кнопку **Select Workbook** и выбрать нужную рабочую книгу (в нашем случае это C:\Fact.xls);
- нажать кнопку **Options**, чтобы открылись дополнительные параметры подключения, и, если вы собираетесь изменять таблицу Excel из программы, снять флажок **Read Only**.

В итоге окно может выглядеть так, как представлено на рис. 9.10.

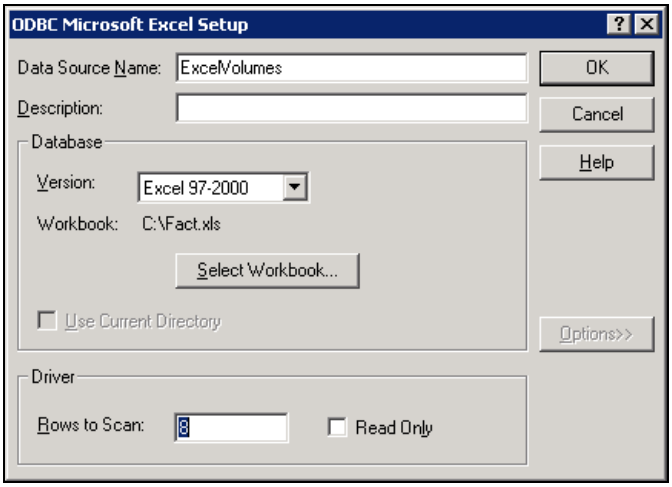


Рис. 9.10. Настроенный источник ODBC для подключения к файлу Excel

Осталось нажать две кнопки **OK**, чтобы закрыть окно создания источника данных ODBC.

В принципе, в коде программы можно написать значение свойства `ConnectionString` вручную, воспользовавшись документацией по ADO. Выглядеть соответствующая строка может, например, так:

```
cn.ConnectionString = "Provider=MSDASQL.1;DSN=FactExcel;" _
    & "DBQ=C:\Fact.xls;"
```

Но зачем писать что-то руками, когда очень просто можно сгенерировать нужное значение автоматически:

- так же, как было описано в предыдущем разделе, создаем UDL-файл (можно воспользоваться уже готовым);
- щелкаем по нему два раза мышью, переходим на вкладку **Provider** и выбираем **Microsoft OLE DB Provider for ODBC Drivers**;
- переходим на вкладку **Connection** и в списке **Use Data Source Name** выбираем созданный нами источник данных **ExcelVolumes**. Остальные поля

можно не заполнять (рис. 9.11). Для проверки можно нажать кнопку **Test Connection**, а затем **OK**.

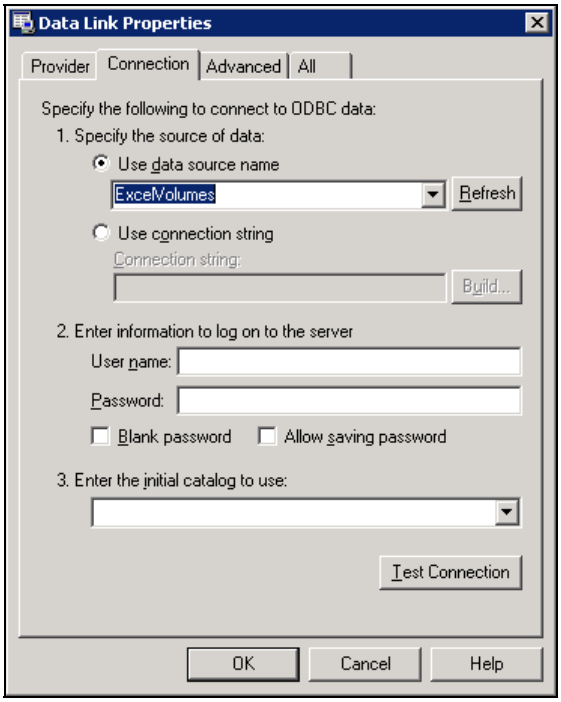


Рис. 9.11. Настройка параметров подключения к созданному источнику ODBC

- последнее действие — открываем созданный UDL-файл в блокноте, копируем из него строку подключения и используем в нашей программе.

Итоговый код процедуры для подключения к Excel может выглядеть так:

```
Public Sub ConnectToExcel()
    Dim cn As New ADODB.Connection
    cn.ConnectionString = "Provider=MSDASQL.1;" _
        & "Data Source=ExcelVolumes"
    cn.Open

    'Про Recordset мы будем говорить в следующем разделе
    'Этот код помещен для наглядной проверки
    Dim rs As New ADODB.Recordset
    rs.Open "Volumes", cn
    MsgBox rs.GetString

End Sub
```

Чтобы подключиться к файлу Excel, нам потребовалось:

- создать именованный диапазон в книге Excel;
- создать источник данных ODBC с именем **ExcelVolumes**;
- написать три строки кода, начиная с создания объекта `Connection` до вызова его метода `Open`.

9.5. Объект *Recordset* и коллекция *Fields*

9.5.1. Открытие *Recordset*

Обычно следующий после установки соединения этап — это создание объекта `Recordset` и работа с ним.

Что такое объект `Recordset`? Само слово `Recordset` расшифровывается как *Set of Records*, т. е. набор записей. Проще всего представить его как таблицу (аналогичную таблицам в Excel), которая находится в оперативной памяти компьютера. Однако у `Recordset` есть принципиальные отличия от таблиц Excel:

- Excel не следит за "строгостью" таблиц. На предприятиях часто можно встретить таблицы, в середину которых вставлены, например, промежуточные итоги по группам или заметки. `Recordset` — это "строгая" таблица. В ней четко определены столбцы и строки и разрывов она не допускает (хотя какие-то значения на пересечении строк и столбцов вполне могут быть пустыми);
- в таблице Excel в одном столбце без проблем можно использовать самые разные значения — числовые, даты и времени, строковые, формулы и т. п. В `Recordset` для каждого столбца определяется тип данных и значения в этом столбце должны соответствовать этому типу данных.

`Recordset` обычно создается на основе данных, полученных с источника (но может быть создан и заполнен вручную), в которых предусмотрены столбцы *(Fields)* и строки *(Rows)*. Создание объекта `Recordset` и заполнение его данными с источника в самом простом варианте выглядит так (подразумевается, что мы открыли при помощи объекта `cn` соединение с учебной базой данных `Northwind` на SQL Server):

```
Dim rs As New ADODB.Recordset
rs.Open "customers", cn
```

Убедиться, что `Recordset` действительно создан и существует, можно, например, при помощи строки:

```
MsgBox rs.GetString
```

При открытии `Recordset` вполне могут возникнуть ошибки, поэтому рекомендуется использовать обработчик ошибок. Специальной коллекции `Errors` в `Recordset` не предусмотрено, а значит, придется обойтись стандартным объектом `Err`.

В нашем примере мы открыли таблицу `Customers` целиком. Однако это не единственный (и не лучший) способ извлечения данных с источника. Для метода `Open()` рекомендуется использовать запрос на языке SQL. Например, в нашем случае можно было бы использовать такой код:

```
rs.Open "select * from dbo.customers", cn
```

Запрос использовать лучше, потому что:

- есть возможность указать фильтр `Where` (условие обязательно заключать в одинарные кавычки) и скачать в `Recordset` не все записи, а только удовлетворяющие вашему условию;
- есть возможность точно так же ограничить число возвращаемых столбцов — снова сокращение объема передаваемых данных и уменьшение расхода памяти;
- есть возможность использовать функции SQL, сортировку на источнике данных и множество полезных дополнительных возможностей.

Очень часто в реальных приложениях текст запроса "склеивается" из кусочков, которые поступают из разных мест. Например, пользователь выбрал в раскрывающемся списке имя заказчика, и для события `Change` этого списка тут же сработала процедура, которая выполнила запрос на SQL Server, получив данные только для этого заказчика, и присвоила полученные значения другим элементам управления. В нашем случае соответствующая строка кода может выглядеть так:

```
rs.Open "select * from dbo.customers Where CompanyName = " & "'" _
        & ComboBox1.Value & "'" , cn
```

Набор символов `"'"` — это одинарная кавычка внутри двух двойных. Такая конструкция нужна, чтобы текст запроса мог выглядеть, например, так:

```
select * from dbo.customers Where CompanyName = 'Alfreds Futterkiste'
```

Причина проста — в языке SQL строковые значения нужно заключать в одинарные кавычки.

Если вы ответственны не только за создание клиентского приложения, но и за проектирование базы данных, бывает очень удобно предусмотреть запрос данных только через представления. Это позволит более гибко управлять системой безопасности и в случае необходимости перестройки базы данных

(например, разбиения главной таблицы на текущую и архивную) сэкономить множество времени.

И еще один практический момент. Конечно, для работы с базами данных знать язык запросов SQL очень полезно. Литературы по нему очень много, и его вполне реально освоить за несколько дней. Однако, если вы — обычный пользователь и не имеете об языке SQL никакого представления, ничего страшного. Просто открывайте таблицы целиком без всяких запросов, а все остальное можно будет сделать средствами VBA.

9.5.2. Настройки курсора и другие параметры открытия *Recordset*

При открытии объекта `Recordset` можно определить еще несколько важных его свойств (их можно определить как напрямую перед открытием, так и передать как дополнительные параметры методу `Open()`).

- Первое свойство — `CursorType`, тип курсора. Это свойство определяется только перед открытием `Recordset` (после открытия оно доступно только для чтения). Курсор можно представить себе как указатель на записи в `Recordset`. В зависимости от типа курсора мы определяем возможности работы с `Recordset` и производительность выполняемых операций (чем больше возможностей, тем меньше производительность, и наоборот). Можно задавать следующие значения:
  - `adOpenForwardOnly` — это значение используется по умолчанию. Оно оптимизировано для достижения максимальной производительности (его возможности будут минимальными). Курсор может двигаться только вперед, а изменения, вносимые другими пользователями, видны не будут;
  - `adOpenStatic` — то же самое, что и предыдущее значение, за исключением того, что курсор может двигаться во всех направлениях;
  - `adOpenKeyset` — позволяет двигаться курсору в любом направлении, видны только изменения существующих записей другими пользователями (удаление старых записей и добавление новых не видны);
  - `adOpenDynamic` — обеспечивает максимальные возможности: позволяет двигаться в любых направлениях, видны любые изменения в записях, производимые другими пользователями. К сожалению, провайдер `Microsoft.Jet.OLEDB.4.0` этот тип курсора не поддерживает, поэтому с Access и Excel его использовать не получится.

  Свойство `Recordset.RecordCount` нормально функционирует только для курсоров типа `adOpenStatic` и `adOpenKeyset`. Для курсоров типа

`adOpenForwardOnly` и `adOpenDynamic` оно возвращает –1, поскольку драйвер подключения не может определить количество записей.

- Второе важное свойство — `CursorLocation`. Оно определяет, где будет создан курсор — на сервере или на клиенте. По умолчанию используется значение `adUseServer` (создавать на сервере). Второе значение — `adUseClient` (создавать на клиенте). В целом, практически во всех ситуациях удобнее и производительнее использовать серверные курсоры, за одним исключением — в реализации серверных курсоров на разных источниках данных много отличий, поэтому если вы планируете работать с разными источниками, имеет смысл подумать о клиентских курсорах.
- Третье важное свойство — `LockType`. Это свойство определяет тип блокировок, которые будут наложены на записи на источнике, помещенные в `Recordset`. Можно использовать следующие значения:
  - `adLockReadOnly` — записи в `Recordset` будут доступны только для чтения, вы не сможете их изменять. Это значение используется по умолчанию;
  - `adLockPessimistic` — наиболее надежный, с точки зрения целостности данных, вид блокировки. Вы можете изменять записи в `Recordset`, но при начале изменения записи она блокируется на источнике таким образом, что другие пользователи не смогут обратиться к ней ни на чтение, ни на запись, пока вы не вызовете методы `Update()` или `CancelUpdate()`;
  - `adLockOptimistic` — это значение позволяет выиграть в производительности за счет проигрыша в надежности обеспечения целостности данных. Запись на источнике блокируется только на время выполнения метода `Update()`. Остальные пользователи могут одновременно с вами читать и изменять данные на источнике;
  - `adLockBatchOptimistic` — то же самое, что и `adLockOptimistic`, но вместо немедленного обновления по одной записи используется пакетное обновление. В ситуации, когда изменяется большое число записей, такое решение позволяет выиграть в производительности.

Первый параметр метода `Open()` в наших примерах был как именем таблицы, так и командой SQL (могут использоваться и другие варианты). Поскольку драйвер OLE DB не знает, чем может быть передаваемый текст, он взаимодействует с сервером баз данных, чтобы определить это. На практике такое выяснение может сильно тормозить работу приложения, поэтому имеет смысл перед открытием `Recordset` явно указать тип передаваемых параметров. Это делается при помощи параметра `Options`, который передается этому методу.

Наиболее часто используемые значения такие:

- `adCmdText` — передается команда SQL;
- `adCmdTable` — передается имя таблицы (равносильно указанию сгенерировать команду SQL, которая вернет все записи из таблицы);
- `adCmdTableDirect` — также передается имя таблицы для того, чтобы получить все ее записи напрямую (без выполнения SQL-запроса), если источник поддерживает такую операцию;
- `adCmdStoredProc` — передается имя хранимой процедуры для ее выполнения, а то, что она вернет, используется для заполнения `Recordset`.

С практической точки зрения важно запомнить следующее. Если вам нужно обеспечить себе возможность перемещения по `Recordset` в любом направлении и изменять в нем записи, код на открытие `Recordset` должен быть таким:

```
Dim rs As New ADODB.Recordset
rs.CursorType = adOpenStatic
rs.LockType = adLockOptimistic
rs.Open ... 'Пишем, что именно мы открываем
```

На практике я пишу строки:

```
rs.CursorType = adOpenStatic
rs.LockType = adLockOptimistic
```

совершенно автоматически, поскольку они нужны в подавляющем большинстве случаев. Какие-то другие значения этих свойств нужно использовать только в специальных ситуациях (например, когда важнее всего производительность).

9.5.3. Перемещение по *Recordset*

После того как объект `Recordset` создан, нам необходимо выполнять с ним различные операции. Самое простое действие, с которого мы начнем, — перемещение по объекту `Recordset`.

В `Recordset` всегда имеется ограниченное количество записей (столько, сколько мы получили с источника). Изначально курсор устанавливается на первую запись в `Recordset` (убедиться в этом можно при помощи свойства `AbsolutePosition`). Однако если мы выполним команду `MovePrevious()` (но только один раз), ошибки не произойдет, а если мы попробуем выполнить команду `MovePrevious()` второй раз, то возникнет ошибка. `AbsolutePosition` вернет загадочное значение –2. Связано это с тем, что в `Recordset` перед первой записью, полученной с источника, помещается специальная запись `BOF`

(от англ. *Begin Of File*, хотя никаких файлов, конечно же, нет). Проверить, находимся ли мы на этой специальной записи, можно при помощи свойства `BOF`. Например, такой код (если он выполнен сразу после открытия `Recordset`):

```
Debug.Print rs.BOF
rs.MovePrevious
Debug.Print rs.BOF
```

вернет нам вначале `False`, а затем `True`.

Точно так же после последней записи в `Recordset` находится специальная запись `EOF` (от *End Of File*). Проверить, не находится ли курсор на ней, можно при помощи аналогичного одноименного свойства `EOF`.

Иногда бывает так, что сразу после открытия `Recordset` и `BOF`, и `EOF` одновременно возвращают `True`. Объяснение такой ситуации очень простое — в `Recordset` с источника по каким-то причинам не вернулось ни одной записи. Рекомендуется во избежание неожиданностей предусматривать сразу после открытия `Recordset` проверку на наличие в нем записей.

После того, как мы определились с текущей позицией в `Recordset`, необходимо разобраться с тем, как можно по нему перемещаться. Проще всего это делать при помощи методов с префиксом `Move...`

- `Move()` — этот метод принимает два параметра: `NumRecords` — на сколько записей необходимо переместиться (это число может быть и отрицательным, что означает переместиться назад) и второй параметр (необязательный) — имя закладки, с которой нужно начать перемещение. Можно использовать три встроенные закладки: для текущей, первой и последней записи. Если имя закладки не указано, то перемещение начинается с текущей позиции.
- `MoveFirst()`, `MoveLast()`, `MoveNext()` и `MovePrevious()` — назначения этих методов понятны из названий: перемещение на первую, последнюю, следующую и предыдущую запись соответственно.

Необходимо отметить, что перемещение назад (при помощи `MovePrevious()` или `Move()` с отрицательным значением) для курсора, открытого как `adOpenForwardOnly`, может привести к совершенно непредсказуемому результату (в зависимости от источника данных) — от ошибки до перехода на случайную запись.

Чаще всего для прохода по всем записям используется такой нехитрый алгоритм:

```
rs.MoveFirst
Do Until rs.EOF
```

```
    'Что-то делаем с каждой записью
    'Например, получаем значение нужного поля
    rs.MoveNext
Loop
```

Если необходимо напрямую перепрыгнуть на нужную запись, можно использовать методы `Find()` и `Seek()`.

- `Find()` — предназначен для поиска по значению одного столбца. Он принимает в качестве параметра критерий поиска, насколько нужно отступить от исходной позиции, направление поиска и откуда нужно начать поиск. Очень удобно, что при определении критерия поиска можно использовать оператор `Like` с подстановочными символами. При обнаружении нужной записи метод `Find()` переставляет курсор на найденную запись, если же запись не обнаружена, то курсор устанавливается на `EOF` (или `BOF`, если поиск был в обратном порядке). Например, чтобы найти все немецкие фирмы в нашем `Recordset` для таблицы `Customers`, можно использовать код вида:

  ```
  rs.Find "country = 'Germany'"
  Do While Not rs.EOF
      Debug.Print "Название фирмы: " & rs.Fields("CompanyName")
      mark = rs.Bookmark
      rs.Find "country = 'Germany'", 1, adSearchForward, mark
  Loop
  ```

  В этом примере используются еще незнакомые нам объекты `Fields` и `Bookmark`, но их назначение понятно: объект `Field` нужен, чтобы вывести название фирмы, а объект `Bookmark` — чтобы продолжить поиск, начиная со следующей записи по отношению к последней найденной.

- `Seek()` — отличается от метода `Find()` тем, что он ищет значение по индексу (объект `Index` для `Recordset` создается либо программным способом, либо автоматически, если на таблицу, на основе который был создан `Recordset`, было наложено ограничение первичного ключа *(Primary Key)*). Этот метод работает только для серверных курсоров с типом команды `TableDirect`, и поэтому к использованию не рекомендуется.

Хочется упомянуть о еще одном свойстве, которое может сильно помочь в перемещении по `Recordset` (и которое уже встречалось в наших примерах) — свойстве `Bookmark`. Это свойство очень простое — достаточно присвоить его значение переменной типа `Variant`, когда указатель стоит в нужном месте `Recordset`, а затем присвоить этому свойству значение этой переменной, чтобы опять на него вернуться, как в нашем примере с поиском.

Вообще говоря, значение, которое возвращает это свойство, изначально совпадает с номером записи в `Recordset`, однако Microsoft честно предупреждает, что таким способом пользоваться закладкой очень не рекомендуется — если курсор стоит в одном и том же месте, свойство `Bookmark` может возвращать разные значения.

9.5.4. Коллекция *Fields* и объекты *Field*

Главное содержание `Recordset` — это то, что лежит в ячейках на пересечении строк (в `Recordset` они называются записями *(records)* и представлены объектами `Record`) и столбцов. В `Recordset` столбцы называются полями и представляются объектами `Field`, которые сведены в коллекцию `Fields`. Объекты `Record` используются нечасто, поскольку имен у них нет, а переходить между записями проще при помощи свойств и методов самого объекта `Recordset` — `AbsolutePosition`, `Find()`, `Move()` и т. п. Коллекция же `Fields` и объекты `Field` используются практически в каждой программе.

У коллекции `Fields` все свойства стандартные, как у каждого объекта `Collection`.

- `Count` — возвращает, сколько всего столбцов в `Recordset`.
- `Item` — позволяет вернуть нужный столбец (объект `Field`) по имени или номеру. Поскольку это свойство является свойством по умолчанию, то можно использовать код, как в нашем примере:

  ```
  rs.Fields("CompanyName")
  ```

  Есть еще один вариант синтаксиса для обращения к этому свойству:

  ```
  rs!CompanyName
  ```

Методы у этой коллекции есть как стандартные, так и специфические.

- `Append()` — добавляет новый столбец в `Recordset`. `Delete()` — удаляет столбец. Обе команды разрешено выполнять только на закрытом `Recordset` (пока не был вызван метод `Open()` или не установлено свойство `ActiveConnection`).
- `Update()` — сохраняет изменения, внесенные в `Recordset`. Метод `CancelUpdate()` — отменяет изменения, внесенные в `Recordset`.
- `Refresh()` — загадочный метод, который ничего не делает (о чем честно написано в документации). Обновить структуру `Recordset` данными с источника можно только методами самого объекта `Recordset`.
- `Resync()` — работает только для коллекции `Fields` объекта `Record` (не `Recordset`), обновляя значения в строке.

Намного больше интересных свойств у объекта Field.

- ActualSize — реальный размер данных для текущей записи, DefinedSize — номинальный размер данных для столбца (в байтах), в соответствии с полученной с источника информацией.
- Attributes — определяет битовую маску для атрибутов столбца (допускает ли пустые значения, можно ли использовать отрицательные значения, можно ли обновлять, используется ли тип данных фиксированной длины и т. п.)
- Name — просто строковое имя столбца. Для столбцов, полученных с источника, это свойство доступно только для чтения.
- NumericScale и Precision — значения, которые определяют соответственно допустимое количество знаков после запятой и общее максимальное количество цифр, которое можно использовать для представления значения.
- Value — самое важное свойство объекта Field. Определяет значение, которое находится в столбце (если мы пришли через коллекцию Fields объекта Record, то значение этой записи Record; если через Fields объекта Recordset — текущей записи). Доступно и для чтения, и для записи (в зависимости от типа указателя). ADO позволяет работать с большими двоичными данными (изображения, документы, архивы), что очень удобно. Свойство OriginalValue возвращает значение, которое было в этом столбце до начала изменений, UnderlyingValue — значение, которое находится на источнике данных (пока мы работали с Recordset, оно могло быть изменено другим пользователем, и поэтому OriginalValue и UnderlyingValue могут не совпадать). Свойство Value — это свойство объекта Field по умолчанию (т. е. то свойство, значение которого будет возвращаться, если не указывать, к какому свойству объекта мы обращаемся), поэтому следующие две строки равноценны:

```
Debug.Print rs.Fields("CompanyName")
Debug.Print rs.Fields("CompanyName").Value
```

- Status — значение этого свойства, отличное от adFieldOK (значение 0), означает, что поле было недавно программно добавлено в Recordset.
- Type — определяет тип данных столбца в соответствии с приведенной в документации таблицей. Например, для типа данных nvarchar возвращается 202.

У объекта Field есть только два метода: AppendChunk() и GetChunk(). Оба эти метода используются только для работы с большими двоичными типами данных (изображениями, документами и т. п.), когда использовать обычными способами свойство Value не получается.

9.5.5. Сортировка и фильтрация данных

Данные в `Recordset` помещаются в том порядке, как они пришли из источника. Если специальный порядок сортировки в запросе не указан, то данные возвращаются в соответствии с параметрами источника данных (например, на SQL Server они будут по умолчанию упорядочены по кластерному индексу, который по умолчанию создается для первичного ключа). Можно произвести сортировку на сервере, указав в запросе выражение `ORDER BY`, а можно выполнить сортировку на клиенте при помощи свойства `Sort` объекта `Recordset`.

Общее правило выглядит так: если есть возможность, всегда нужно выполнять сортировку на сервере. Сервер намного лучше оптимизирован для выполнения подобных операций, на нем обычно больше оперативной памяти и процессорных ресурсов. На клиенте сортировку выполнять следует только тогда, когда это невозможно сделать на сервере (например, вы получаете данные от хранимой процедуры, в тексте которой сортировка не предусмотрена). Однако, в принципе, сортировка в `Recordset` менее ресурсоемка, чем могла бы быть (данные физически не перемещаются, только создается новый индекс для поля, по которому производится сортировка), поэтому ее вполне можно использовать в программах даже для большого количества данных.

Применение свойства `Sort` связано с двумя серьезными ограничениями:

- его можно использовать только тогда, когда курсор открыт на клиенте (т. е. для свойства `CursorLocation` должно быть установлено значение `adUseClient`, по умолчанию он открывается на сервере);
- это свойство нельзя использовать с некоторыми драйверами, например, нельзя сортировать данные при подключении к Excel.

Применение этого свойства выглядит очень просто:

```
rs.Sort = "CompanyName"
```

При этом то, что передается этому свойству, должно быть названием столбца (т. е. именем объекта `Field`) в `Recordset`.

Можно передавать несколько названий столбцов и разные порядки сортировки:

```
rs.Sort = "Country ASC, CompanyName DESC"
```

`Recordset` вначале будет отсортирован по стране (`Country`), а потом — по имени компании (`CompanyName`) в убывающем порядке `DESC` (по умолчанию используется `ASC` — по возрастанию, поэтому в нашем примере это слово можно опустить).

Чтобы отменить сортировку (и вернуться к записям в том порядке, в котором они были возвращены с источника), достаточно присвоить этому свойству пустое значение:

```
rs.Sort = ""
```

В Recordset записи можно фильтровать. Отфильтрованные записи остаются в Recordset, но являются невидимыми при выполнении операций перемещения и поиска, и курсор на отфильтрованных записях установить нельзя.

Для фильтрации используется свойство Filter объекта Recordset. Оно может принимать три типа значений:

- строковое значение, по синтаксису аналогичное передаваемому методу Find():

  ```
  rs.Filter = "LastName = 'Smith' AND FirstName = 'John'"
  ```

  Можно использовать оператор Like с подстановочными символами (только '*' и '%'). Отличие от Find() заключается в том, что в Find() просто устанавливается курсор на первую найденную подходящую запись, а в Filter все неподходящие становятся невидимыми;
- массив закладок;
- несколько специальных значений — все конфликтующие записи, записи, ожидающие сохранения на источнике, и т. п.

Снять фильтрацию можно точно так же, как и сортировку:

```
rs.Filter = ""
```

9.5.6. Изменение записей на источнике при помощи объекта *Recordset*

Очень часто возникает необходимость из приложения не только получить информацию о записях с источника, но и внести на источник изменения. При этом обычно возникает множество сложностей, связанных с решением вопроса о том, в какой таблице данные изменять (если у нас набор таблиц), с блокировками, производительностью, разрешениями, каскадными обновлениями, возможностью отката внесенных пользователем изменений и т. п. Многие проблемы решаются намного проще, если вы изначально будете следовать правилу: любые изменения можно проводить только при помощи хранимых процедур (и, соответственно, при помощи объекта Command). Далее будут рассмотрены возможности внесения изменений через объект Recordset, которые следует использовать только в самых простых случаях.

Необходимо также помнить, что значение свойства `LockType` при открытии `Recordset` по умолчанию устанавливается в `adLockReadOnly`, что не позволяет вносить изменения в `Recordset`. Вам потребуется изменить значение этого свойства перед открытием `Recordset`.

Общая схема внесений изменений через `Recordset` выглядит так: вначале вызывается один из нужных нам методов (`AddNew()`, `Edit()`, `Delete()`) и производится внесение изменений в оперативной памяти на клиенте. Следующая операция — вызов метода `Update()` для `Recordset`, который и производит запись внесенных изменений на источник данных (после вызова `Delete()` вызывать метод `Update()` не нужно). Далее приведена информация о каждом из этих методов.

- Работа с методом `AddNew()` — это всегда операция, которая состоит из трех частей:
  - вначале нужно вызвать этот метод, чтобы создать новую пустую запись (курсор будет установлен на нее автоматически);
  - занести значения в столбцы, используя свойство `Value` коллекции `Fields`;
  - вызвать метод `Update()` для записи изменений на источник.

  Пример использования этого метода к нашему `Recordset` может выглядеть так:

```
Dim rs As ADODB.Recordset
Set rs = CreateObject("ADODB.Recordset")
rs.CursorType = adOpenKeyset
rs.LockType = adLockOptimistic
rs.Open "select * from dbo.customers", cn
rs.AddNew
rs.Fields("CompanyName") = "Test rs company"
rs.Fields("Country") = "Germany"
rs.Fields("CustomerId") = "TESTR"
rs.Update
```

  В принципе, можно присваивать новые значения и в самом методе `AddNew()`, но с точки зрения синтаксиса это сложнее.

- Изменение существующей записи при помощи метода `Edit()` выглядит очень просто:

```
rs.Find "CustomerID='ALFKI'"
rs.Fields("ContactName") = "Маша"
rs.Update
```

- Удаление записи с помощью метода `Delete()` — еще проще:

```
rs.Find "CustomerID='TESTR'"
rs.Delete
```

Обратите внимание, что в этом случае метод `Update()` вызывать не нужно!

Конечно, при внесении изменений на источник ошибки могут возникнуть по множеству причин. Рекомендуется всегда в таких ситуациях устанавливать обработчик ошибок и анализировать свойства стандартного объекта `Err`.

9.5.7. Прочие свойства и методы объекта *Recordset*

Далее рассказывается о некоторых дополнительных свойствах и методах объекта `Recordset`, которые используются намного реже.

- `AbsolutePage`, `PageSize`, `PageCount` — эти свойства позволяют использовать группы записей (страницы) для перемещения по `Recordset`. По умолчанию размер страницы равен 10 записям.
- `ActiveCommand` — позволяет вернуть объект `Command`, представляющий команду, которая использовалась на источнике при создании `Recordset` и заполнении его записями. Подробнее про объект `Command` будет рассказано в *разд. 9.6*.
- `ActiveConnection` — возвращает объект `Connection`, который использовался для создания `Recordset`. Передать (или получить) строковое значение, на основе которого будет создан объект `Connection`, можно при помощи свойства `Source`.
- `CacheSize` — позволяет определить количество записей, которые будут находиться в оперативной памяти на клиенте (остальные записи будут подкачиваться по мере необходимости с источника). Используется тогда, когда количество записей в `Recordset` очень большое или приходится работать с записями очень большого размера, например, с большими двоичными объектами.
- `EditMode` — позволяет определить состояние текущей записи: не изменялась, изменялась, но изменения еще не переданы на источник, удалена и т. п.
- `InsertCommand`, `DeleteCommand`, `UpdateCommand` — позволяют определить объекты `Command`, представляющие команды, которые будут использоваться на источнике при создании, удалении и изменении записей в `Recordset` соответственно.

- `MarshalOptions` — позволяет определить, какие записи при изменении `Recordset` будут возвращаться с клиента на сервер: все (по умолчанию) или только измененные.
- `MaxRecords` — это свойство настоятельно рекомендуется указывать перед открытием для всех `Recordset`, для которых существует возможность получить с источника очень большое количество записей (что может привести к нехватке оперативной памяти на клиенте). Оно определяет максимальное количество записей, которые могут быть скачаны в `Recordset`. Вместо этого свойства можно использовать и `CacheSize`.
- `State` — позволяет определить, что в настоящее время происходит с `Recordset`. Используется одно из 5 значений: открыт, закрыт, соединяется, выполняет команду на источнике или получает оттуда данные.
- `Status` — позволяет определить результат последней операции обновления данных.

Методы объекта `Recordset` представлены в следующем списке.

- `Cancel()` — позволяет прервать открытие `Recordset` (например, если оно затянулось).
- `CancelBatch()` и `CancelUpdate()` — позволяют отменить внесенные в `Recordset` изменения (до вызова команды `Update()`) в разных режимах.
- `Clone()` — позволяет скопировать объект `Recordset` в другой объект `Recordset` со всеми закладками (обычно используется, когда нужно иметь более чем одну текущую запись).
- `Close()` — позволяет освободить память, занимаемую данными `Recordset` (но не удаляет сам объект). В случае необходимости вы можете опять вызвать метод `Open()`, чтобы воссоздать этот объект с ранее определенными параметрами (значения свойств объекта `Recordset` при вызове метода `Close()` сохраняются).
- `CompareBookmarks()` — позволяет сравнить две закладки и вернуть результат сравнения (указывают на одну и ту же запись, или первая запись выше, или первая запись ниже и т. п.).
- `GetRows()` — позволяет вернуть из `Recordset` двумерный массив типов `Variant`. В качестве необязательных параметров принимает стартовую позицию, количество строк, которые нужно поместить в массив, и те столбцы, которые нужно извлечь из `Recordset`. В нашем примере использование этого метода может выглядеть так:

```
Dim arr As Variant   'объявляем переменную для создаваемого массива
arr = rs.GetRows
Debug.Print arr(2, 3)
```

- `GetString()` — самая простая возможность получить из объекта `Recordset` строковое значение. По умолчанию разделителями между столбцами принимаются символы табуляции, между записями — перевод каретки. Можно использовать в отладочных целях, чтобы посмотреть на `Recordset`:

  ```
  Debug.Print rs.GetString
  ```

- `NextRecordSet()` — позволяет очистить текущий `Recordset` и выполнить следующую команду, указанную в методе `Open()`, если команды были указаны в формате:

  ```
  "select * from dbo.customers; select * from dbo.employees"
  ```

  Обычно такой подход используется для обработки наборов однотипных таблиц.

- `Requery()` — повторно выполняет запрос, который использовался для метода `Open()`, и заново заполняет `Recordset`.

- `Resync()` — обновляет значения уже полученных записей, скачав их заново с источника. Новые записи при этом видны не будут (в отличие от метода `Requery()`).

- `Save()` — сохраняет `Recordset` в файле на диске. Можно использовать формат ADTG (Microsoft Advanced Data TableGram), XML или родной формат провайдера. Например:

  ```
  rs.Save "C:\rscustomers.xml", adPersistXML
  ```

  При необходимости можно восстановить сохраненный `Recordset` из файла, указав соответствующие параметры методу `Open()`.

- `SetAllRowsStatus()` — позволяет изменить значение свойства `Status` для всех строк `Recordset`.

- `Supports()` — выполняет проверку того, что поддерживает данный `Recordset` (в каких направлениях можно двигаться, поддерживается ли поиск, закладки и т. п.). Те возможности, которые проверяются этим методом, определяются особенностями провайдера (т. е. драйвера для подключения к источнику).

Для объекта `Recordset` предусмотрен также набор событий (`EndOfRecordset`, `FetchComplete`, `MoveComplete`), но используются они редко, поэтому здесь рассматриваться не будут.

9.6. Объект *Command* и коллекция *Parameters*

В самых простых случаях, когда можно получать и изменять данные напрямую в таблицах, можно обойтись объектом `Recordset`. Однако во многих си-

туациях возможностей этого объекта недостаточно. Как уже говорилось ранее, предпочтительнее производить любое внесение изменений на источник данных при помощи хранимых процедур. Часто существует потребность в создании временных таблиц и других объектов на сервере. Бизнес-логика многих приложений (начисление процентов, абонентской платы, формирование специальных отчетов с вычислениями и т. п.) также реализована в виде хранимых процедур, поэтому в реальных приложениях одним объектом `Recordset` не обойтись.

Для выполнения команд SQL на сервере (в том числе для запуска хранимых процедур, команд DDL для создания объектов, выполнения служебных операций типа резервного копирования, восстановления, изменения параметров работы) необходимо использовать объект `Command`.

Создание этого объекта производится очень просто:

```
Dim cmd As ADODB.Command
Set cmd = CreateObject("ADODB.Command")
```

Следующее, что нужно сделать, — это назначить объекту `Command` объект подключения `Connection`. Для этой цели предназначено свойство `Command.ActiveConnection`. Ему можно передать готовый объект `Connection`, а можно сформировать этот объект неявно, используя в качестве значения свойства `ActiveConnection` строку подключения. Рекомендуется всегда предавать готовый объект подключения: во-первых, так для соединения можно настроить больше параметров, а во-вторых, если вы используете в приложении несколько объектов `Command`, можно использовать для каждого такого объекта одно-единственное подключение, что экономит ресурсы. В нашем примере мы используем созданный нами ранее объект `Connection`:

```
cmd.ActiveConnection = cn
```

Следующая наша задача — выбрать тип команды. В принципе, для многих источников можно этого не делать — модули ADO постараются сами выяснить у источника данных, что это за команда (хранимая процедура, SQL-запрос и т. п.), однако лучше всегда его определять: экономится время и системные ресурсы, уменьшается вероятность ошибок. Для выбора типа команды используется свойство `CommandType`. Значения, которые ему можно присвоить, аналогичны возможным значениям параметра `Options` метода `Open()` объекта `Recordset`, которое было рассмотрено в *разд. 9.5.2*. Например, если мы передаем команду на выполнение хранимой процедуры, то присвоить соответствующее значение можно так:

```
cmd.CommandType = adCmdStoredProc
```

Следующее действие — определить текст команды, которая будет выполняться. Делается это при помощи свойства `CommandText`. Например, если мы

хотим запустить на выполнение хранимую процедуру `CustOrderHist`, то соответствующий код может выглядеть так:

```
cmd.CommandText = "CustOrderHist"
```

Чаще всего хранимая процедура требует передачи ей одного или нескольких параметров. Делается это при помощи коллекции `Parameters` и объектов `Parameter`. Для определения параметров можно использовать два способа:

- вначале создать объекты `Parameter` автоматически путем запроса к серверу (используется метод `Refresh()` коллекции `Parameters`), а затем присвоить им значения:

```
cmd.Parameters.Refresh
cmd.Parameters(1) = "ALFKI"
```

- создать объекты `Parameter` вручную и вручную добавить их в коллекцию `Parameters`. Этот способ более экономичен (нет необходимости лишний раз обращаться на сервер), но требует предварительно знать точные свойства параметра и использовать код большего размера:

```
Dim Prm As ADODB.Parameter
Set Prm = cmd.CreateParameter("CustomerID", adVarWChar, _
          adParamInput, 5, "ALFKI")
cmd.Parameters.Append Prm
```

После определения параметров команду необходимо запустить на выполнение. Для этого используется метод `Execute()`. Самый простой способ его вызова выглядит так:

```
cmd.Execute
```

Этот метод принимает также три необязательных параметра, при помощи которых можно дополнительно определить параметры, тип вызываемой команды и т. п.

Некоторые хранимые процедуры и передаваемые команды не требуют возврата каких-либо значений (кроме кода ошибки), но это бывает редко. Как же принять значение, возвращаемое выполняемой командой?

Если возвращаемое значение официально зарегистрировано как возвращаемый параметр (например, оно помечено ключевым словом `OUT` в определении хранимой процедуры), то это значение будет присвоено соответствующему параметру объекта `Command`, и до него можно будет добраться обычным способом — при помощи свойства `Value`.

Если же, как в нашем примере с `CustOrderHist`, возвращаемое значение просто сбрасывается в поток вывода (в нашем случае возвращается набор записей), то можно использовать два способа:

- первый способ — использовать то, что метод `Execute()` возвращает объект `Recordset`, заполненный полученными при помощи команды записями:

```
Dim rs2 As ADODB.Recordset
Set rs2 = cmd.Execute()
Debug.Print rs2.GetString
```

- второй способ — воспользоваться тем, что метод `Open()` объекта `Recordset` может принимать в качестве параметра объект `Command` (в этом случае объект `Connection` передавать этому методу уже нельзя):

```
Dim rs2 As ADODB.Recordset
Set rs2 = CreateObject("ADODB.Recordset")
rs2.Open cmd
Debug.Print rs2.GetString
```

Рассмотрим некоторые другие свойства и методы объекта `Command`.

- `CommandStream` — позволяет вместо прямого назначения текста команды (через свойство `CommandText`) принять значение из потока ввода (например, из текстового файла или другой программы). Допустимый формат потока зависит от провайдера (драйвера для данного подключения). Использовать одновременно `CommandStream` и `CommandText` нельзя (второе свойство автоматически становится пустым).
- `CommandTimeout` — позволяет указать, сколько времени в секундах ждать результата выполнения команды на источнике, прежде чем вернуть ошибку.
- `Dialect` — это свойство позволяет указать особенности разбора *(parsing)* текста команды на провайдере.
- `NamedParameters` — определяет, будут ли передаваться провайдеру имена параметров (`True`) или будет использоваться простая передача значений по порядку (`False`, по умолчанию).
- `Prepared` — свойство, которое может влиять на производительность. Если установить его в `True` (по умолчанию `False`), то при первом выполнении команды провайдер создаст ее откомпилированную версию, которую и будет использовать при последующих выполнениях. Первый раз команда будет выполнять медленнее, чем обычно, зато в следующие разы — быстрее. Такое "приготовление команды" *(command preparation)* поддерживают далеко не все драйверы подключений.
- `State` — возвращает те же значения и используется в тех же целях, что и для объекта `Recordset`.
- `Cancel()` — этот метод позволяет прекратить выполнение команды (когда выполнение затянулось), если такую возможность поддерживает провайдер.

Задание для самостоятельной работы 9: Вставка по запросу записей из базы данных

Примечание:

В этом задании используется база данных Microsoft Access с именем Борей.mdb, которая при установке Microsoft Office 2003 по умолчанию помещается в каталог C:\Program Files\Microsoft Office\OFFICE11\SAMPLES. Перед началом этой работы рекомендуется провести поиск на диске, чтобы найти этот файл. Если он находится в другом каталоге — скорректируйте путь к этому файлу в ответе. Если вы его вообще не обнаружите, то доустановите Microsoft Office таким образом, чтобы он был установлен с полным набором компонентов.

Подготовка:

1. Создайте новый документ Word и сохраните его как C:\ InsertQueryResults.doc.
2. Щелкните правой кнопкой мыши по любой панели инструментов или меню и в открывшемся списке доступных панелей инструментов выберите **Элементы управления**.
3. Нажмите кнопку **Режим конструктора** на панели инструментов **Элементы управления** (верхняя левая кнопка) и в этом режиме поместите в документ Word новую кнопку. Для этого нужно щелкнуть по объекту **Кнопка** на панели инструментов **Элементы управления** и в документе определить местонахождение и размеры этой кнопки.
4. Щелкните по созданной вами кнопке правой кнопкой мыши и в контекстном меню выберите **Свойства**. Определите для кнопки свойства по вашему желанию. Выглядеть документ в итоге может, например, так, как показано на рис. 9.12.
5. При помощи меню **Вставка | Закладка** поместите под эту кнопку закладку с именем **Bookmark1**.
6. В режиме конструктора щелкните по кнопке правой кнопкой мыши и в контекстном меню выберите **Исходный текст**. Откроется редактор кода Visual Basic с созданной процедурой для события `Click` данной кнопки. Поместите в него следующий код:

```
Private Sub CommandButton1_Click()
    Dim nEmpId As Integer
    Dim sLastName As String
    Dim sFirstName As String
```

```
    Dim sTitle As String

    nEmpId = CInt(InputBox("Введите номер сотрудника:"))

    'Код, который нужно заменить
    sLastName = "Иванов"
    sFirstName = "Иван"
    sTitle = "Начальник"
    'Конец кода, который нужно заменить

    ThisDocument.Activate
    ThisDocument.Bookmarks("Bookmark1").Select
    Selection.TypeText CStr(nEmpId) & " " & sLastName & " " & _
        sFirstName & " " & vbTab & sTitle & vbCrLf

End Sub
```

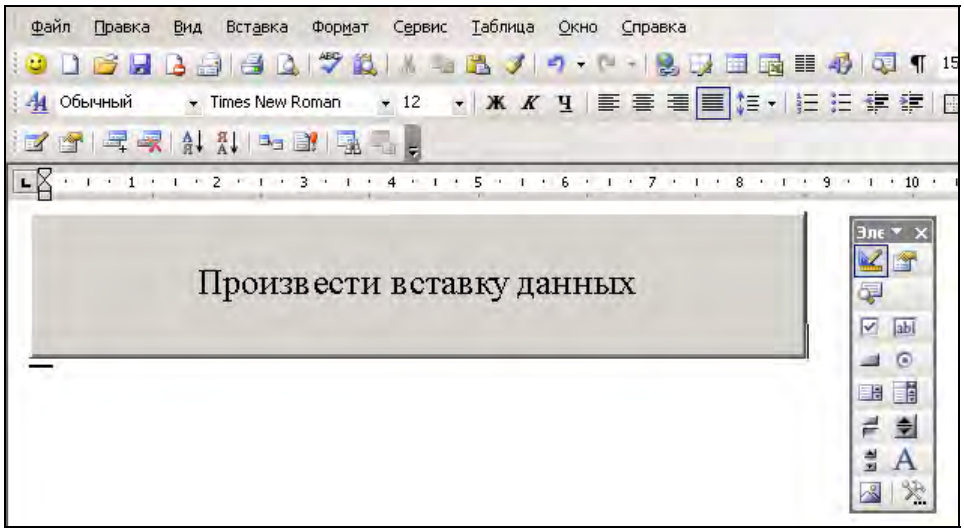


Рис. 9.12. Кнопка в документе Word

ЗАДАНИЕ:

Измените код события `Click` этой кнопки таким образом, чтобы вместо присвоения переменным явно определенных значений им присваивались значения из таблицы `Сотрудники` базы данных Борей.mdb:

- для переменной `sLastName` — значение из столбца `Фамилия`;
- для переменной `sFirstName` — значение из столбца `Имя`;
- для переменной `sTitle` — значение из столбца `Должность`.

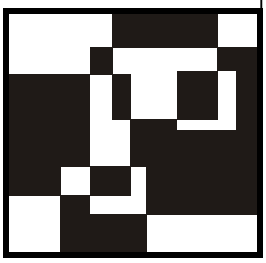
Номер сотрудника (значение столбца `Код сотрудника`) нужной записи должен определяться пользователем при помощи функции `InputBox()`.

Ответ к заданию 9

Итоговый код для события `Click()` нашей кнопки может быть таким:

```
Private Sub CommandButton1_Click()
    Dim nEmpId As Integer
    Dim sLastName As String
    Dim sFirstName As String
    Dim sTitle As String

    'Получаем от пользователя номер сотрудника
    nEmpId = CInt(InputBox("Введите номер сотрудника:"))

    'Создаем и настраиваем объект Connection
    Dim cn As New ADODB.Connection
    'У вас путь к файлу Борей.mdb может быть другим
    cn.ConnectionString = "Provider=Microsoft.Jet.OLEDB.4.0;" & _
        "Data Source=C:\Program Files\Microsoft Office\OFFICE11\" _
        & "SAMPLES\Борей.mdb"
    cn.Open

    'Создаем и настраиваем объект Recordset
    Dim rs As New ADODB.Recordset
    'Обеспечиваем возможность передвижения в любом направлении
    rs.CursorType = adOpenStatic

    'Открываем Recordset на основе запроса
    rs.Open "SELECT [КодСотрудника], [Имя], [Фамилия], [Должность] " & _
        "FROM [Сотрудники] WHERE [КодСотрудника] = " & nEmpId, cn

    'Проверяем, не пустой ли Recordset
    If rs.EOF = True And rs.BOF = True Then
        MsgBox "Сотрудник с таким номером не обнаружен"
        Exit Sub
    End If

    'Присваиваем значения из найденной записи в таблице
    sLastName = rs.Fields("Фамилия")
    sFirstName = rs.Fields("Имя")
    sTitle = rs.Fields("Должность")

    ThisDocument.Activate
    ThisDocument.Bookmarks("Bookmark1").Select
    Selection.TypeText nEmpId & " " & sLastName & " " & sFirstName & _
        " " & vbTab & sTitle & vbCrLf

End Sub
```

ГЛАВА 10



Программирование в Word

10.1. Зачем программировать в Word

Word — старейшее и самое популярное приложение, входящее в состав Microsoft Office. В большинстве организаций пользователи готовят документы именно в Word.

С точки зрения программирования Word — это, прежде всего, средство для изготовления отчетов к базам данных. При этом отчет — это любой документ, который формируется на основе информации из базы данных, например: договор, акт приемки-передачи, приходный кассовый ордер, объявление на взнос наличными, распоряжение в бухгалтерию, накладная и т. п. Конечно же, к отчетам, которые можно формировать в Word, относятся и документы со сводными данными — отчеты за период, ведомости и т. п.

Автору приходилось создавать приложения с отчетами, разработанными в самых разных программных продуктах — Microsoft Access, Crystal Reports, Microsoft Reporting Services и т. д. Если ваше приложение генерирует отчеты к базам данных в Microsoft Word, то, скорее всего, такие отчеты будут не самыми быстрыми, с точки зрения их формирования, и не самыми простыми, с точки зрения программирования. Зато совершенно точно они будут самыми дружелюбными по отношению к конечному пользователю. Почему?

Очень часто на предприятии возникает необходимость исправить в форме отчета всего пару строк — например, вместо "Директор" поставить "И. О. Директора". Если отчет создан в Crystal Reports или в Microsoft Reporting Services, придется срочно обращаться к разработчику. А через какое-то время И. О. утвердят в должности директора, и разработчику придется править отчет снова.

Если же отчет изначально создается в документе Word, то пользователь всегда может сам внести в созданный документ необходимые изменения — по-

давляющее большинство пользователей на предприятии умеют работать в Word. Срочно разыскивать разработчика уже не нужно.

У Word есть и другие преимущества. Как правило, при изготовлении отчетов в Word значения из базы данных подставляются в шаблон отчета, который хранится в базе данных или в файле (с расширением dot). Если формат отчета сложный, с большим количеством специфического оформления (например, объявление на взнос наличными), то намного проще подготовить его шаблон в Word, чем, к примеру, в Crystal Reports или Reporting Services.

Еще одно программное применение Word — умение работать с разными форматами документов. Эту возможность Word вполне можно использовать для массовой обработки документов.

Приведу случай из практики: в каталоге на диске у нас собралось несколько сотен "разнокалиберных" документов разных пользователей. Часть из них создана в Word разных версий, часть — просто текстовые файлы, некоторые документы в форматах HTML, XML или EML (сообщения электронной почты). На предприятии внедрена система документооборота на основе SharePoint Portal Server и нам необходимо привести все эти документы к единому формату (Word 2003) и загрузить их на SharePoint Portal Server. Конечно же, без автоматизации в такой ситуации возиться придется очень долго.

Третье программное применение Word — форматирование документов, например: программное применение стилей, поиск и замена участков текста сразу во многих документах, работа со структурой документа и т. п. Обычно такие задачи ставятся в издательствах, например, при подготовке рукописей.

10.2. Введение в программирование в Word. Объектная модель

Общая структура объектов Word выглядит так, как показано на рис. 10.1.

Но пугаться не стоит — большая часть из этих сотен объектов никогда вам не понадобится. На практике для решения большинства программных задач достаточно знать всего лишь пять объектов (с сопутствующими коллекциями):

- объект `Application`;
- объект `Document` (с коллекцией `Documents`);
- объект `Selection`;
- объект `Range`;
- объект `Bookmark` (с коллекцией `Bookmarks`).

Далее все эти важные объекты будут подробно описаны. Для каждого объекта вначале будут рассмотрены общие моменты, связанные с ними, например,

в каких ситуациях они нужны и как с их помощью выполнять те или иные действия. Поскольку наиболее часто встречающаяся задача программирования в Word — это создание документа (на основе шаблона) и запись в нужное место документа необходимой информации, то акцент будет сделан на использовании соответствующих объектов для решения именно этой задачи.

Кроме того, для каждого объекта будут перечислены самые важные свойства, методы и события с кратким их описанием. Эта часть добавлена по просьбе слушателей учебного курса по программированию в Office, поскольку многие из них не владеют английским настолько, чтобы свободно пользоваться документацией. Если вы читаете эту книгу подряд, то эти справочные части можно просто пропускать.

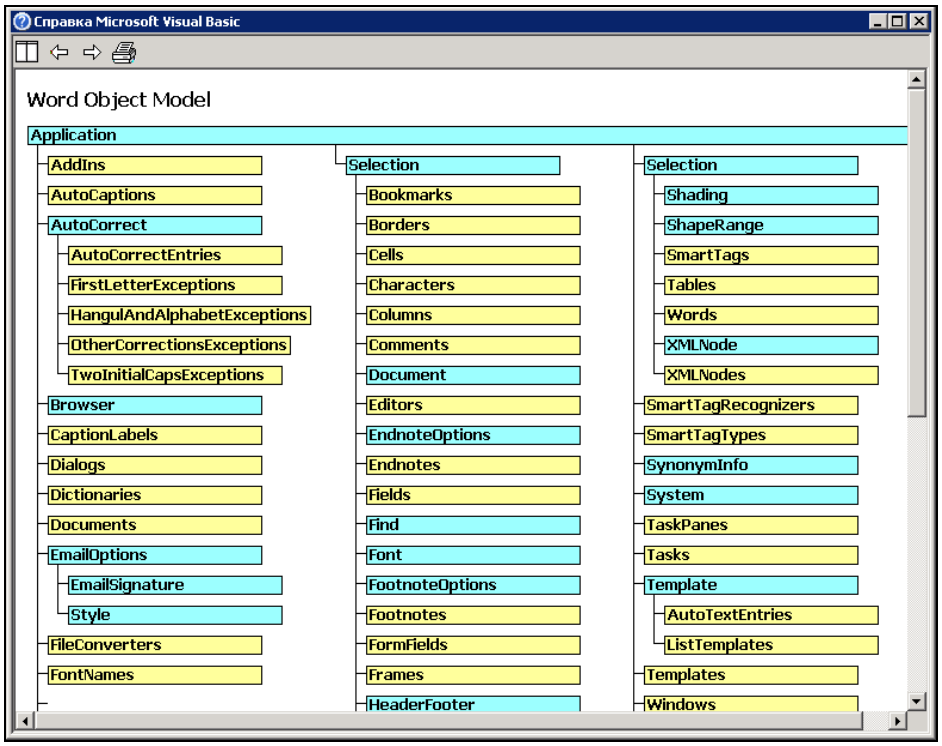


Рис. 10.1. Главные объекты Word

10.3. Объект *Application*

10.3.1. Как работать с объектом *Application*

Объект `Application` — это само приложение Microsoft Word. Все остальные объекты Word "вложены" в него. Создать этот объект — значит, запустить

Word на вашем компьютере. Как правило, это нам и необходимо (если мы создаем документ в формате Word из другого приложения, например из Access). Но не забудьте — если вы запускаете Word из другого приложения Office, то необходимо добавить в ваш проект ссылку на библиотеку Microsoft Word 11.0 Object Library.

Код запуска Word очень прост:

```
Dim oWord As New Word.Application
```

Однако, выполнив его из другого приложения, вы, скорее всего, даже не заметите, что у вас что-то произошло. Причины просты:

1. По умолчанию Word запускается в скрытом окне.
2. Если в нем не открыт ни один документ, он тут же закрывается (после того, как завершается создавшая его процедура).

Сделать Word видимым очень просто:

```
oWord.Visible = True
```

Однако может возникнуть вопрос: а нужно ли его делать видимым? Некоторые разработчики утверждают, что не нужно. Пусть Word работает в скрытом окне, создавая требуемый документ. Когда пользователю потребуется, он этот документ откроет. Как поступать в конкретном случае, решать вам, но я предпочитаю, чтобы Word все-таки был видимым: во-первых, сразу видны все проблемы при создании документа, а во-вторых, пользователям почему-то очень нравится, когда у них на глазах открывается Word и начинает печатать строки, которые в противном случае пришлось бы печатать им самим.

Если вы работаете с Word в скрытом окне, не забудьте после выполнения необходимых действий его закрыть (иначе он так и останется в оперативной памяти, видимый только через **Task Manager**). Для закрытия Word нужно вызвать его метод `Quit()`.

Чтобы Word не закрывался сам собой, в нем достаточно создать новый документ. Подробно об этом будет рассказано в следующем разделе, но самый простой вариант создания нового документа Word выглядит так:

```
Dim oWord As New Word.Application
oWord.Visible = True
oWord.Documents.Add
```

Если Word уже был открыт на компьютере, то можно получить на него ссылку, например, при помощи такого кода:

```
Set oWord = GetObject(, "Word.Application")
```

Однако на практике, кроме очень специальных случаев (активизация объектов OLE), такой подход по сравнению с открытием нового экземпляра Word

ничего не дает. Наоборот, появляется дополнительный риск нечаянно испортить открытый в существующем экземпляре созданный пользователем документ или закрыть его без сохранения пользовательских документов. Поэтому лучше создавать новый экземпляр Word.

Если же ваш код VBA выполняется в Word (т. е. Word уже запущен), объект `Application` создавать не надо. В этой ситуации он будет автоматически доступен в любой момент (чтобы в этом убедиться, достаточно впечатать в окне редактора кода `Application` и добавить точку). Более того, если не указано, к какому объекту относится то или иное свойство или метод, компилятор VBA в Word автоматически считает, что это свойство или метод принадлежит объекту `Application`. Поэтому следующие два фрагмента кода функционально одинаковые:

```
Application.Selection.TypeText "Мой текст"
```

и

```
Selection.TypeText "Мой текст"
```

Еще один важный момент, который связан с объектом `Application` в Word. Для него предусмотрено большое количество удобных в использовании событий (открытие документа, выход из Word, щелчок правой кнопкой мыши, изменение документа, печать документа, сохранение документа и т. п.) Однако по умолчанию все эти события не видны. Чтобы они появились, необходимо в разделе `Declarations` кода формы (а не модуля!) объявить объект `Application` с ключевым словом `WithEvents`, например так:

```
Public WithEvents App As Word.Application
```

В списке объектов у вас появится новый объект `App` (т. е. `Application`), для которого можно выбрать события и добавлять код в событийные процедуры точно так же, как мы это делали для формы и элементов управления.

10.3.2. Свойства, методы и события объекта *Application*

Далее для справки приведены самые важные свойства, методы и события объекта `Application`.

- `ActiveDocument` — возвращает объект активного документа в данном экземпляре Word. Это свойство используется очень активно, обычно без упоминания объекта `Application`, например:

  ```
  ActiveDocument.Save
  ```

  Свойство доступно только для чтения, поэтому чтобы сделать какой-нибудь документ активным, придется вызывать для его объекта метод `Activate()`.

- `ActivePrinter` — позволяет получить или настроить активный принтер в ходе работы программы. Также используется очень активно, например, если результаты работы вашего приложения необходимо печатать на определенном сетевом принтере. Свойство доступно как для чтения, так и для записи.
- `AutomationSecurity` — определяет уровень безопасности при программном открытии файлов. По умолчанию установлено значение `msoAutomationSecurityLow` — открывать с включенными макросами. Можно также использовать значения `msoAutomationSecurityForceDisable` (отключить макросы) и `msoAutomationSecurityByUI` (то, что настроено на графическом интерфейсе).
- `BackgroundPrintingStatus` — определяет, сколько заданий Word стоит в очереди на печать.
- `Browser` — свойство, которое возвращает объект `Browser` (малозаметный набор из трех кнопок, который прячется под вертикальной полосой прокрутки). С программной точки зрения интересно его свойство `Target`, которое может принимать одно из 12 значений (комментарий, сноска, таблица, рисунок, заголовок, страница и т. п.). Затем при помощи методов `Next()` и `Previous()` для этого объекта мы можем перемещаться между этими элементами.
- `Build` — возвращает версию и номер сборки Word. Очень полезно для проверки на совместимость, если ваше приложение работает только под определенными версиями Word.
- `CapsLock` — позволяет проверить, включен ли режим CapsLock на клавиатуре. Изменить этот режим при помощи данного свойства нельзя (только для чтения), для этого есть другие средства (связанные с использованием Windows API). Аналогично работает свойство `NumLock`.
- `Caption` — позволяет заменить текст "Microsoft Word" в заголовке окна на другой, например "Мое приложение".
- `CheckLanguage` — возвращает `True`, если Word определяет в автоматическом режиме язык, на котором производится ввод текста. Если в системе установлено несколько языков ввода, то по умолчанию автоматическая проверка установлена. При помощи этого свойства можно изменить режим работы Word.
- `COMAdIns` — позволяет получить ссылку на коллекцию загруженных COM Ad ins — встраиваемых в Word приложений, построенных по технологии COM. Очень удобно использовать перед обращением к данному встраиваемому приложению.

- `CustomizationContext` — свойство, которое позволяет указать шаблон или документ, на который будут распространяться внесенные вами изменения в меню, панели инструментов и клавиатурные комбинации. Например, код:

  ```
  CustomizationContext = NormalTemplate
  ```

  говорит о том, что все изменения, которые вы будете вносить, начиная с этого момента, будут сохраняться в шаблоне Normal.dot (и, таким образом, будут применяться ко всем документам).

- `Dialogs` — возвращает коллекцию `Dialogs`, представляющую из себя все возможные диалоговые окна Word. При помощи этой "ветви" объектной модели Word вы можете открыть любое из сотен диалоговых окон Word и определить действия, которые будут предприняты при выборе пользователем тех или иных параметров в данном диалоговом окне. К сожалению, эта "ветвь" очень плохо документирована, и при использовании объектов диалоговых окон приходится заниматься самостоятельными исследованиями (при помощи макрорекордера и окна **Locals**), чтобы определить нужные свойства и их значения. По моему опыту, обычно бывает проще создать свою форму VBA, которая будет выполнять необходимые действия, чем заниматься такой исследовательской работой. Пример использования диалогового окна открытия файла может выглядеть так:

  ```
  Dim oDlg As Dialog
  Set oDlg = Application.Dialogs(wdDialogFileOpen)
  If oDlg.Display = -1 Then
      MsgBox "Вы выбрали файл: " & _
          Application.Options.DefaultFilePath(wdCurrentFolderPath) & _
          "\" & oDlg.Name
  End If
  ```

  Для диалоговых окон, которые предназначены для работы с файлами, в объекте `Application` предусмотрено отдельное свойство `FileDialog`, возвращающее одноименный объект.

- `DefaultSaveFormat` — определяет формат сохранения файлов Word по умолчанию (тот, который будет предлагаться пользователь в диалоговом окне **Save As**). Можно настроить на сохранение в формате обычного текста TXT, текста Unicode, RTF и т. п.

- `DisplayAlerts` — очень важное свойство. Оно позволяет подавить вывод ошибок и диалоговых окон при работе макросов и приложений VBA. Во многих ситуациях без него не обойтись. Особенно часто прибегать к этому свойству требуется, когда в ходе работы программы необходимо что-нибудь удалить или закрыть без сохранения.

- `DisplayAutoCompleteTips` — включает или отключает подсказки для автозавершения текста. Чаще всего необходимо отключить.
- `Documents` — самое важное свойство. Возвращает коллекцию документов. Подробнее про эту коллекцию и работу с документами читайте в *разд. 10.4*.
- `EmailOptions` — возвращает очень сложный и насыщенный свойствами объект `EmailOptions`, который используется для настройки Word как редактора почтовых сообщений Outlook.
- `EnableCancelKey` — это свойство определяет, сможет ли пользователь прервать выполнение любого макроса при нажатии клавиш <Ctrl>+<Break>. Если установить для этого свойства значение `wdCancelDisabled`, то это приведет к тому, что макрос, вошедший в бесконечный цикл, можно будет закрыть только вместе с Word — через Task Manager.
- `FeatureInstall` — еще одно свойство, которое позволяет не раздражать пользователя попытками Office доустановить еще не установленные компоненты. Для этого нужно установить это свойство в значение `msoFeatureInstallNone`.
- `FileDialog` — возвращает объект `FileDialog`, т. е. окно выбора файла, каталога, открытия файла или сохранения. Для открытия этого окна необходимо воспользоваться методом `Show()` этого объекта.
- `FileSearch` — возвращает объект `FileSearch`, который может использоваться для поиска файлов по определенным параметрам.
- `International` — еще одно очень важное свойство. Возвращает информацию о текущих региональных настройках даты, времени, валюты, отображения чисел, локализации версии Word и т. п.
- `IsObjectValid` — очень удобное свойство для всевозможных проверок (открыт ли документ, находится ли указатель в таблице и т. п.). Проверяет, существует ли еще объект, к которому мы хотим обратиться. Позволяет уберечь от ошибок, когда, например, документ или объект в документе был удален пользователем.
- `KeyBindings` — очень удобное во многих ситуациях свойство. Оно возвращает коллекцию `KeyBindings` — привязок клавиатурных комбинаций. Проще говоря, при помощи этого объекта и его подобъектов вы можете назначить любую команду Word или любой макрос любому сочетанию клавиш (в том числе и тем, которые уже заняты служебными командами, например <Alt>+<F4>). Общая последовательность действий при этом выглядит так:

- определяем свойство `CustomizationContext` объекта `Application`, т. е. где будут сохраняться наши изменения: в шаблоне Normal.dot, в текущем документе или в шаблоне, прикрепленном к текущему документу;
- при помощи метода `Application.BuildKeyCode()` определяем цифровой код для нашей клавиатурной комбинации;
- при помощи метода `KeyBindings.Add()` добавляем новое назначение, при этом определяем все необходимые параметры.

Например, чтобы по нажатию клавиш <Alt>+<D> у нас запускался макрос `DataLoad()` во всех документах, можно выполнить следующий код:

```
CustomizationContext = NormalTemplate
Application.KeyBindings.Add wdKeyCategoryMacro, _
    "Normal.NewMacros.DataLoad", BuildKeyCode(wdKeyAlt, wdKeyD)
```

- `Language` — еще одно свойство, которое позволяет определить, установлена ли на компьютере пользователя локализованная версия Word (точнее, это свойство определяет язык пользовательского интерфейса). Для русского языка будет возвращаться значение 1049, для английского — 1033. Более подробную информацию (о языке помощи, языке программы установки и т. п.) можно получить при помощи свойства `LanguageSettings`.
- `MacroContainer` — очень полезное свойство для программистов. Позволяет в ходе выполнения макроса определить, откуда был запущен текущий программный код (обычно проверяются два варианта — Normal.dot или текущий документ).
- `NewDocument` — одна из возможностей создать новый документ Word. Возвращает объект `NewDocument`. Для создания нового документа используется метод `Application.NewDocument.Add()`.
- `NormalTemplate` — это свойство позволяет получить ссылку на объект `Template`, представляющий Normal.dot, для внесения в него изменений.
- `Option` — возвращает объект `Option` с огромным количеством свойств. Через этот объект программным способом можно настроить значения на всех вкладках окна **Параметры** (меню **Сервис | Параметры**).
- `Path` — возвращает путь к программным файлам Word на диске.
- `PrintPreview` — с помощью этого свойства можно перейти в режим предварительного просмотра текущего документа или проверить, находимся ли мы в этом режиме. Очень удобно для показа документа пользователю или для реализации своей процедуры печати.
- `ScreenUpdating` — свойство, которое позволяет запретить перерисовку экрана (если установить его значение в `False`). Обычно используется для ускорения работы процедур, которые выводят что-то на экран.

- `Selection` — еще одно важнейшее свойство. Возвращает объект `Selection` — то место, в котором находится указатель вставки. Подробнее о нем — в *разд. 10.5*.
- `ShowStartupDialog` — определяет, показывать или нет **Task Panel** (панель задач в правой части документа) при запуске Word. Чаще всего используется для отключения показа. Есть еще несколько свойств с префиксом `Show...`, значения которых очевидны.
- `SpecialMode` — позволяет проверить, не находится ли Word в специальном режиме копирования и вставки (для перехода в этот режим нужно выделить текст и нажать <F2> или <Shift>+<F2>, а потом переместить курсор и нажать <Enter>).
- `StartUpPath` — предоставляет возможность просмотреть/определить путь к каталогу автозапуска. Те шаблоны и встраиваемые приложения, которые находятся в этом каталоге, Word при запуске открывает автоматически. По умолчанию каталог автозапуска находится в профиле пользователя. Путь к нему выглядит как **\application data\microsoft\word\startup**.
- `StatusBar` — еще одно очень полезное свойство. Позволяет вывести текст в Status Bar (строка состояния), т. е. в строке в нижней части окна приложения, где выводится информация о страницах, столбцах, языке, режимах работы и т. п.
- `System` — возвращает одноименный объект `System`, предназначенный для получения информации из операционной системы (региональный настройки, тип курсора мыши, разрешение экрана, тип процессора и т. п.). Позволяет также подключать сетевые диски и запускать приложение Microsoft System Information.
- `Tasks` — возвращает одноименную коллекцию `Tasks` с объектами `Task`, представляющими все работающие в системе процессы. При помощи этих объектов можно программным способом найти работающее в системе приложение и что-нибудь с ним сделать (сделать видимым или невидимым, активизировать, закрыть, передать в его окно сообщение Windows, как при работе с Windows API и т. п.). Опытные разработчики активно используют этот набор объектов для работы с внешними приложениями. Запускать внешние приложения лучше всего при помощи специального объекта `Shell`, о котором будет рассказано в *разд. 10.6.10*.
- `UserControl` — очень важное свойство (оно есть и в Excel). Это свойство позволяет определить, как именно был запущен Word — пользователем вручную или программным образом. На основе этого можно, например, сделать вывод, нужно ли его программным образом закрывать.

- `UserInitials` и `UserName` — позволяет получить или определить информацию об инициалах или имени пользователя. Инициалы используются в исправлениях, а имя пользователя — в свойствах документа.
- `VBE` — это свойство возвращает недокументированный, но очень интересный объект `VBE`, представляющий редактор Visual Basic. Обычно используется для программного внесения изменений в проекты VBA, например, добавление ссылок.
- `Version` — свойство возвращает версию Word (менее подробную, чем свойство `Build`). Для Word 2003 значение этого свойства равно 11.0.
- `Visible` — позволяет спрятать окно Microsoft Word очень качественно — Word исчезает и с рабочего стола, и из панели задач.
- `Windows` — возвращает информацию об одноименной коллекции `Windows`, содержащей объекты `Window`. Эти объекты представляют окна документов Word.
- `WindowsState` — позволяет свернуть, развернуть или восстановить окно Word.

Самые важные методы объекта `Application` приведены далее.

- `Activate()` — просто активизирует окно Word с текущим документом. Обычно нужно активизировать определенный документ, поэтому этот метод используется для объекта `Document`.
- `BuildKeyCode()` — позволяет узнать уникальный номер для клавиатурной комбинации в Word. Пример использования этого метода был приведен ранее при рассмотрении свойства `Application.KeyBindings`.
- `ChangeFileOpenDirectory()` — этот метод позволяет изменить каталог, который по умолчанию открывает Word при работе с документами (по умолчанию задан, конечно, каталог **Мои документы**);
- `CheckGrammar()` и `CheckSpelling()` — позволяют проверить грамматику и орфографию для передаваемых символьных значений. Чаще всего используются аналогичные методы для объектов `Document` и `Range`.
- `CleanString()` — очень полезный метод. Позволяет "чистить" передаваемое символьное значение (полученное, например, от объектов `Selection` или `Range`) от специальных символов Word и превращает их в обычный текст, как будто он был набран в блокноте.
- `DefaultWebOptions()` — возвращает одноименный объект, при помощи которого можно определить множество свойств, используемых при сохранении документа Word в формате HTML (кодировка, работа с изображениями, CSS, с какими браузерами обеспечивать совместимость и т. п.).

- `GoBack()` — этот метод обеспечивает переход на последнее место редактирования в документе. Word сохраняет с документом три последние точки редактирования, так что открыть последний документ в Word и перейти на точку, где вы остановились, можно очень просто:

```
RecentFiles(1).Open
Application.GoBack
```

- `GoForward()` — обеспечивает переход вперед по точкам сохранения.
- `KeyBoard()` — очень полезный метод. Позволяет программным способом переключать раскладку клавиатуры в Word, уберегая таким образом пользователей от ошибок. Переключение на русский язык выглядит так:

```
Application.Keyboard 1049
```

  а на английский:

```
Application.Keyboard 1033
```

  Если этому методу ничего не передавать, он вернет текущую раскладку клавиатуры.

- `KeyString()` — метод, обратный `BuildKeyCode()`. Если `BuildKeyCode()` возвращает уникальный идентификатор клавиатурной комбинации, то этот метод возвращает клавиатурную комбинацию для данного уникального идентификатора.
- `ListCommands()` — метод, не похожий на другие. Он создает новый документ и выводит в нем в виде таблицы справочник по методам и клавиатурным комбинациям Word, как стандартным, так и назначенным вами.
- `OnTime()` — очень интересный метод. Он позволяет выполнить макрос Word либо в указанное вами время, либо по прошествии какого-то времени. В Word одновременно может работать только один таймер. При помощи этого метода можно выполнять ресурсоемкие операции в автоматическом режиме.
- `OrganizerCopy()` — еще один полезный метод. Позволяет скопировать макрос, панель инструментов, запись автотекста или стиль из одного документа в другой. Для удаления и переименования этих элементов предусмотрены методы `OrganizerDelete()` и `OrganizerRename()`.
- `PrintOut()` — метод, который принимает огромное количество параметров (все необязательные) и позволяет вывести на печать весь документ или его часть. Может использоваться для объектов `Application`, `Document` и `Window`.
- `Quit()` — метод, который используется, видимо, чаще всех. Позволяет закрыть Word с сохранением или без сохранения документов.

- `Repeat()` — просто повторяет последнюю выполненную команду указанное вами количество раз.
- `ResetIgnoreAll()` — снять метку со всех фрагментов текста, помеченных как "без проверки" в ходе проверки орфографии.
- `Run()` — еще один очень важный метод. Позволяет запустить процедуру/макрос из открытого шаблона или документа с передачей параметров.
- `ScreenRefresh()` — обновляет окно приложения. Обычно используется после того, как автоматическое обновление было отключено при помощи свойства `ScreenUpdating`.
- `ShowClipboard()` — отображает панель буфера обмена Word (если вы работаете с несколькими буферами).

Остальные методы относятся к работе протокола DDE, преобразованию различных единиц измерений и т. п.

У объекта `Application` есть множество событий: открытие, закрытие, сохранение и печать документа, щелчки мышью, активизация, выход из приложения и т. п. Единственное, что следует еще раз отметить — события объекта `Application` по умолчанию не отображаются в редакторе Visual Basic. Чтобы они появились, в раздел `Declarations` нужно поместить следующую строку кода:

```
Public WithEvents App As Word.Application
```

В этом случае в списке объектов в окне редактора кода для форм появится объект `App` со всеми необходимыми событиями.

10.4. Коллекция *Documents* и объекты *Document*

10.4.1. Как работать с коллекцией *Documents*

На одну ступень ниже объекта `Application` в объектной модели Word (и по логике использования в приложениях) находятся коллекция `Documents` и объекты `Document`, из которых она состоит. При программировании в Word без них обычно не обойтись.

Чаще всего в программах нам нужно:

1. Запустить Word.
2. Создать или открыть документ.
3. Что-то с этим документом сделать (например, впечатать в нужные места этого документа значения, полученные из базы данных или от пользователя).

Запуск Word производится при помощи объекта `Application`, с которым вы уже знакомы. Для выполнения различных действий с документом используются объекты `Selection`, `Range` и `Bookmark`, которые будут рассмотрены в *разд. 10.5*. А вот второй пункт — создание или открытие документа, проверка, открыт уже документ или нет, сохранение документа и т. п. — реализуется при помощи коллекции `Documents` и объекта `Document`.

Самый простой вариант создания документа выглядит так:

```
Dim oDoc As Word.Document
Set oDoc = Application.Documents.Add()
```

При этом мы создали обычный пустой документ (на основе шаблона Normal.dot) и получили ссылку на него в объектную переменную `oDoc`. Далее в документ можно программно вводить нужную нам информацию.

Однако создавать пустой документ и формировать все его содержимое удобно лишь тогда, когда документ очень простой. Попробуйте программно создать какой-нибудь документ посложнее, например, объявление на взнос наличными или большой договор. Возможно, у вас это получится, но работы придется выполнить очень много. Намного проще набрать и оформить сложный документ обычным способом, как простой документ Word, и оставить в нем пустые места для заполнения из программы. Проще всего это сделать при помощи шаблона Word.

Например, в нашей ситуации нам вначале нужно набрать текст договора, оставив пустые места для изменяемых данных, и сохранить его с расширением dot (предположим, что он сохранен на диске C: с именем dog_blank.dot). Тогда создать документ на основе этого шаблона можно так:

```
Dim oDoc As Word.Document
Set oDoc = Application.Documents.Add("C:\dog_blank.dot")
```

А далее при помощи объектов `Bookmark` и `Range` *(см. разд. 10.5)* вводим текст в оставленные пустые места.

Шаблоны документов можно хранить в разных местах:

- первый вариант — просто *в файле на диске локального компьютера пользователя*. Это самый простой вариант, но у него есть недостатки: во-первых, пользователь может случайно его изменить, а во-вторых, удобнее использовать общий централизованный набор шаблонов для всех пользователей на предприятии, не копируя их на компьютер каждого пользователя;
- второй вариант — *хранить шаблоны в скрытом сетевом каталоге на файл-сервере*, доступном только на чтение. При этом нам не придется заботиться о наличии необходимого шаблона на компьютере пользователя.

Однако и есть здесь проблемы: программа получается неавтономной, зависящей от внешних файлов на файл-сервере. Перенос ее, например, в филиалы будет сопряжен со сложностями;

- третий вариант — *поместить шаблон в базу данных*. Удобнее всего поместить шаблон в виде объекта OLE в базу данных Access. В этой же базе данных Access удобно разместить код приложения, графические формы для пользователя и т. п. Запуск Word при этом будет производиться программно из Access. Конечно, такая программа сможет обращаться не только к данным в базе данных Access, но и к данным на внешних источниках: на SQL Server, Oracle и т. п.

Третий вариант наиболее удобен, поскольку ваше приложение со всеми шаблонами, программным кодом и т. п. будет состоять из единственного файла mdb, а документы будут создаваться как обычные файлы Word. Кроме того, сам по себе Access — это очень мощное приложение, в нем есть множество удобных возможностей. Единственная проблема в этом случае — то, что метод `Documents.Add()` согласен принимать шаблон только в виде файла на диске, а про базы данных он ничего не знает. Можно предварительно извлекать шаблон документа из базы данных и сохранять его в файле во временном каталоге, но лучше идти другим путем и активизировать объект шаблона в базе данных методом `OLEActivate()`. При этом у нас автоматически будет запущен Word, а в нем будет создан новый документ на основе шаблона из базы данных. Подробнее о том, как это делается — в *гл. 12*.

Часто возникает потребность программным способом не создавать новый документ, а открыть уже имеющийся и что-то сделать с ним. Открыть документ проще всего при помощи метода `Open()` коллекции `Documents`. Самый простой вариант применения этого метода выглядит так:

```
Dim oDoc1 As Word.Document
Set oDoc1 = Documents.Open("c:\doc1.doc")
```

Если документ уже открыт, то по умолчанию просто создается ссылка на этот открытый документ, вместо открытия его заново.

Важная (и неочевидная) особенность метода `Open()` заключается в том, что при его использовании во время открытия файла не открывается диалоговое окно **Предупреждение системы безопасности**, в котором пользователь может отключить макросы. За счет этого разработчик может гарантировать работоспособность своего приложения или реализовать систему защиты — от печати, доступа к разным функциям и т. п.

Сохранять документы лучше при помощи методов `Save()` и `SaveAs()` объекта `Document`. В коллекции `Documents` есть также свой метод `Save()`, который позволяет сохранить сразу все открытые документы Word, но обычно это менее удобно.

Заметим, что из Word можно открывать не только документы Word разных версий, но и документы десятков других различных форматов, про которые знает Word — TXT, HTML, XML и т. п. Сохранять файлы также можно в одном из десятков встроенных форматов или использовать свой собственный пользовательский формат (при помощи объекта `FileConvertor`). За счет этого программным образом при помощи макросов можно очень удобно преобразовывать большое количество документов, которые могут находиться, например, в разных каталогах на файловых серверах, или в общих папках Exchange Server, или в базе данных SharePoint Portal Server. Для прохода по дереву каталогов удобнее всего использовать объект `FileSystemObject` из библиотеки Microsoft Scripting Runtime, которая есть на любом компьютере Windows *(см. гл. 4)*.

10.4.2. Свойства и методы коллекции *Documents*

Коллекция `Documents`, как уже говорилось ранее, представляет все документы Word, открытые в настоящий момент. Нумерация документов в коллекции начинается с 1. Из свойств этой коллекции интерес может представлять только свойство `Count` — количество открытых документов. Гораздо важнее методы коллекции `Documents`. Про некоторые из них мы уже говорили в предыдущем разделе, но здесь для справки приведем информацию о них еще раз.

- `Add()` — этот метод позволяет создать и сразу же открыть новый документ (и вернуть ссылку на его объект). Это наиболее распространенный способ создания новых документов в Word. Полный синтаксис этого метода выглядит как:

  ```
  Add(Template, NewTemplate, DocumentType, Visible)
  ```

  Здесь `Template` — это шаблон для создания нового документа, `NewTemplate` (`True/False`) — делать ли новый документ шаблоном, `DocumentType` — тип документа, может принимать значения: `wdNewBlankDocument`, `wdNewEmailMessage`, `wdNewFrameset` или `wdNewWebPage` (по умолчанию новый чистый документ), `Visible` — будет ли новый документ видимым (по умолчанию) или невидимым. Все эти параметры являются необязательными. Если не указать ни один из них, будет просто создан новый документ на основе шаблона Normal.dot (как будто вы создали новый документ при помощи меню **Файл | Создать**).

- `Open()` — еще один важнейший метод коллекции `Documents`. Позволяет открыть документ с диска и добавить его в коллекцию. Этот метод принимает множество параметров, из которых обязательным является только один — имя документа (вместе с путем к нему). Самый простой вариант применения этого метода выглядит так:

```
Dim oDoc1 As Document
Set oDoc1 = Documents.Open("c:\doc1.doc")
```

- `Item()` — позволяет найти нужный документ в коллекции по его индексу. Обычно для получения ссылки на нужный документ используется конструкция `For...Next` с проверкой значения какого-либо свойства документа через `If...Then`. Чаще всего это свойство — `Name`:

```
Dim oDoc1 As Word.Document
For i = 1 To Documents.Count
    Set oDoc1 = Documents.Item(i)
    If oDoc1.Name = "doc1.doc" Then
        Exit For
    End If
    Set oDoc1 = Nothing
Next
```

  Этот код возвращает ссылку в виде переменной `oDoc1` на документ doc1.doc, если он есть в коллекции. Если его нет, то во избежание ошибок нужно реализовывать дополнительные проверки. На практике можно было бы перед сравнением привести имя документа в нижний регистр, если учитывать регистр букв при поиске вам не нужно.

  Через метод `Item()` можно получить доступ к объекту документа напрямую. Например, в этом примере мы получаем имя первого документа в коллекции `Documents`:

```
MsgBox Documents.Item(1).Name
```

- `Save()` и `Close()` — позволяют соответственно сохранить или закрыть все документы в коллекции.
- `CanCheckOut()` (можно ли "забрать" документ в монопольный доступ) и `CheckOut()` (забрать документ в монопольный доступ) — эти методы можно применять, если документ находится в документной библиотеке в базе данных SharePoint Portal Server.

10.4.3. Работа с объектом *Document*, его свойства и методы

После того, как мы при помощи объекта `Application` запустили Word, при помощи коллекции `Documents` создали (или открыли, или нашли среди уже открытых) нужный нам документ, можно выполнять с этим документом различные действия, реализованные при помощи свойств, методов и событий объекта `Document`. У этого объекта десятки свойств и методов, и здесь мы рассмотрим только наиболее важные и часто используемые.

Обратите внимание, что к объекту `Document` можно обращаться и не создавая специальную объектную переменную. Существует еще, по крайней мере, три способа получения доступа к объекту `Document`:

- работать с документом как с элементом коллекции `Documents`. Формат обращения может выглядеть, например, как `Documents.Item(1)`;
- использовать специальное ключевое слово `ThisDocument`. При помощи него можно получить ссылку на объект документа, которому принадлежит исполняемый программный модуль, например:

  ```
  MsgBox ThisDocument.Name
  ```

- использовать свойство `ActiveDocument` объекта `Application`. Это свойство возвращает нам объект активного документа:

  ```
  MsgBox Application.ActiveDocument.Name
  ```

  или просто

  ```
  MsgBox ActiveDocument.Name
  ```

Самые важные свойства объекта `Document` представлены далее.

- `ActiveWritingStyle` — текущий активный стиль (заголовок определенного уровня, обычный текст, гиперссылка и т. п.). Рекомендуется проверять это свойство перед вводом текста.
- `AttachedTemplate` — предоставляет возможность подключить шаблон (со всеми макросами, стилями, записями автотекста и т. п.) или проверить, какой шаблон подключен (вручную это можно сделать через меню **Сервис | Шаблоны и надстройки**).
- `Background` — возвращает объект `Shape`, представляющий фоновый рисунок (фоновые рисунки видны только в режиме Web-документ).
- `BuiltInDocumentProperties` — позволяет получить ссылку на коллекцию `DocumentProperties` с одноименными объектами, представляющими встроенные свойства документа (название, автор, категория, комментарии и т. п.);
- `Characters` — возвращает коллекцию объектов `Range`, каждый из которых представляет один символ. Это свойство есть не только у объекта `Document`, но и у объектов `Selection` и `Range`. Может использоваться, например, для выполнения операция поиска и замены или статистических подсчетов (например, если переводчику платят за количество символов);
- `Content` — свойство, возвращающее объект `Range`, представляющий собой главную цепочку документа *(main document story)*. Если говорить проще, то это просто текст документа, без колонтитулов, сносок, комментариев и т. п.

- `CustomDocumentsProperty` — свойство, возвращающее коллекцию объектов `DocumentProperties`, представляющих пользовательские свойства документа. Можно использовать для сохранения вместе с документом любых значений переменных. Очень удобно, например, для подсчета количества открытий документа, флажков печатался/не печатался, сколько раз вызывалась та или иная функция, на каких компьютерах и каким пользователем открывался и т. п.
- `DefaultTabStop` — определяет отступ по умолчанию при использовании символа табуляции. По умолчанию задано 35 пунктов, что равно примерно 1,25 см.
- `DisableFeatures` — отключает возможности, которые понимают только последние версии Word (для совместимости с пользователями, у которых на компьютерах стоят старые версии). Обычно само свойство `DisableFeatures` просто включает такой режим, а конкретный уровень совместимости задается при помощи свойства `DisableFeaturesIntroducedAfter`.
- `DoNotEmbedSystemFonts` — позволяет не вставлять в документ системные шрифты (по умолчанию вставляются для русского, японского и других языков с набором символов, отличным от латиницы). Позволяет сократить размер документа, но тогда пользователи в системе, где не установлен русский язык, не смогут прочесть этот документ.
- `EmbedTrueTypeFonts` — очень полезное свойство, если вы работаете с документом в месте, где используются экзотические шрифты (например, в издательстве). Вставка шрифтов true-type гарантирует, что получатели документа будут видеть его точно таким же, как и создатель.
- `Envelope` — позволяет получить ссылку на специальный объект `Envelope`, который используется для создания почтовых конвертов.
- `Fields` — позволяет получить ссылку на коллекцию `Fields` одноименных объектов. Это свойство очень полезно при работе с полями. Поле в Word — это место в документе, отведенное для подстановки изменяемых данных: формул, даты, информации об авторе, размере документа и т. п. При работе с документом Word средствами обычного графического интерфейса добавить новое поле можно при помощи меню **Вставка | Поле**.
- `Footnotes` — возвращает коллекцию сносок.
- Свойства с префиксом `Formatting...` — определяют, что показывать в списке стилей панели инструментов **Форматирование**.
- `FormFields` — аналогично `Fields`, но в этом случае возвращается ссылка на поля в формах.
- `FullName` — возвращает полное имя объекта (вместе с путем к нему в файловой системе или Web). Доступно только для чтения.

- `GrammarChecked` — помечает весь документ, как проверенный с точки зрения грамматики (фактически отключает проверку грамматики для данного документа). Такое же свойство существует и у объекта `Range`. Коллекцию ошибок, выловленных при проверке грамматики, можно получить при помощи свойства `GrammaticalErrors`, а выделить ошибки зеленым волнистым подчеркиванием (если этого не сделано) — при помощи свойства `ShowGrammaticalErrors`. Для орфографических ошибок существуют аналогичные свойства `SpellingChecked`, `SpellingErrors` и `ShowSpellingErrors`.
- `HasPassword` — проверяет, назначен ли пароль для указанного документа. Другое свойство `Password` назначает пароль. По причине крайней слабости защиты пароли в Word, Excel и Access использовать не рекомендуется.
- `Indexes` — возвращает коллекцию индексов (т. е. предметных указателей) для документа.
- `Name` — имя документа (без пути к нему).
- `OpenEncoding` — возвращает кодовую страницу, которая использовалась для открытия документа. Для русского языка по умолчанию это 1251.
- `PageSetup` — позволяет получить ссылку на одноименный объект. Используется в основном при реализации печати.
- `Paragraphs` — возвращает ссылку на коллекцию абзацев в данном документе.
- `Path` — возвращает путь к документу в файловой системе (без имени). Это свойство может пригодиться, чтобы создать еще один файл в том же каталоге.
- `Permission` — позволяет получить доступ к объекту `Permission`, который управляет системой внутренних разрешений документа Word (но не разрешений файловой системы).
- `PrintRevisions` — определяет печатать или нет пометки редактора (исправления) вместе с документом. По умолчанию — печатать.
- `ProtectionType` — проверяет защиту данного документа (разрешено все, или только комментарии, чтение, изменения в полях форм и т. п.). Сама защита устанавливается при помощи метода `Protect()`.
- `ReadOnly` — определяет, можно ли вносить изменения в документ или он доступен только для чтения. Это свойство само доступно только для чтения, поскольку соответствующий атрибут устанавливается в файловой системе.
- `RemoveDateAndTime` и `RemovePersonalInformation` — удаляют информацию о дате и времени произведенных изменений и всю информацию о пользова-

теле из документа (включая свойства документа). Могут быть полезными при создании файла-образца.

- `Saved` — очень важное свойство. Позволяет определить, изменялся ли документ со времени последнего сохранения.
- `SaveEncoding` — позволяет явно указать (или получить) кодировку, которая будет использоваться при сохранении документа.
- `SaveFormat` — позволяет получить информацию о формате документа (DOC, RTF, TXT, HTML и т. п.). Доступно только для чтения.
- `Sections` — возвращает коллекцию разделов документа. `Sentences` — то же самое для предложений. Аналогично работают свойства `Shapes`, `Styles`, `Subdocuments`, `Tables`, `Windows` и `Words`.
- `Type` — возвращает тип документа (обычный, шаблон или Web-страница с фреймами).
- `Variables` — еще одно очень удобное свойство. Можно использовать для сохранения своих служебных данных вместе с документом, как и пользовательские атрибуты *(custom attributes)*, но, в отличие от пользовательских атрибутов документа, пользователям эти свойства не будут видны.

Теперь приведу самые важные методы объекта `Document`.

- `Activate()` — этот метод позволяет сделать указанный вами документ активным (например, для ввода текста).
- `AddToFavorities()` — позволяет добавить ссылку на документ в каталог **Избранное**. Может быть полезным, если пользователь будет работать с этим документом постоянно.
- `CheckSpelling()` и `CheckGrammar()` — запускают проверку орфографии и грамматики соответственно.
- `Close()` — закрывает документ. Можно закрыть с сохранением (по умолчанию), а можно без (если указать соответствующий параметр).
- `Compare()` — сравнивает документ с другим и генерирует редакторские пометки в местах, где обнаружены различия.
- `DetectLanguage()` — определяет язык текста. Проверка производится по предложениям, на основе сверки слов со встроенными словарями. Такая проверка производится автоматически во время ввода текста или при открытии нового документа. Чтобы заново провести проверку языков, свойство `LanguageDetected` нужно перевести в `False`.
- `FitToPages()` — очень интересный метод. Размер шрифта автоматически меняется таким образом, чтобы текст стал занимать на одну страницу меньше. Можно использовать для устранения "висячих страниц" и других проблем верстки.

- `FollowHyperlink()` — открывает указанный вами документ в соответствующем приложении (если HTML, то в Internet Explorer).
- `GoTo()` — очень мощный метод. Для объектов `Document` и `Range` он возвращает объект `Range`, для объекта `Selection` — просто перемещает указатель ввода текста на нужное место. Возвращаемые объекты в зависимости от параметров, которые были переданы этому методу, могут указывать на начало страницы, на определенные строки, закладки, комментарии, таблицы, секции, поля, ссылки, формулы и т. п. Может переходить на определенный номер такого элемента в документе, первый, последний, следующий и т. п. Очень удобно использовать для установки указателя в нужное место для автоматического ввода текста.
- `Merge()` — позволяет произвести слияние двух документов. Метод очень сложный и мощный, основывается на применении редакторских пометок.
- `PresentIt()` — открывает данный документ в PowerPoint.
- `PrintOut()` — очень сложный метод, который позволяет вывести на печать весь документ или его часть.
- `PrintPreview()` — переводит документ в режим предварительного просмотра.
- `Protect()` — ограничивает внесения изменений в документ при помощи пароля или нового средства управления правами на доступ к данным, которое называется IRM. Те же возможности на графическом экране доступны через меню **Файл | Разрешения**.
- `Range()` — очень важный метод. Возвращает объект `Range` *(см. разд. 10.5)*, принимает в качестве параметров номер начального символа диапазона и номер конечного символа.
- `Redo()` — повторяет последнее действие. В качестве параметра принимает количество последних действий, возвращает `True`, если повтор был произведен успешно.
- `Repaginate()` — выполняет переразбивку документа на страницы. Обычно используется, если автоматическая разбивка была ранее отключена (например, на вкладке **Общие** диалогового окна **Параметры** (меню **Сервис | Параметры**) или программно при помощи объекта `Options`).
- `Save()` — смысл этого метода очевиден. Если документ ранее не сохранялся, открывается диалоговое окно **Сохранить как**.
- `SaveAs()` — очень мощный и сложный метод. Можно определить путь для сохраняемого документа, его формат, кодировку, пароли на открытие и изменение документа, вставку шрифтов и многое другое. Очень удобно использовать, например, для автоматической конвертации документов.

- `Select()` — позволяет просто выделить весь документ. Этот метод существует для очень большого количества объектов, в том числе для `Selection` и `Range`.
- `TransformDocument()` — исключительно мощный метод, но только для программистов, которые хорошо разбираются в XML и XSLT. Позволяет применить к документу таблицу преобразований стилей (Extensible Stylesheet Language Transformation, XSLT), при помощи которой можно поменять все, что угодно.
- `Undo()` — отменяет определенное количество последних действий. По синтаксису и принципам работы — полный аналог метода `Redo()`.
- `UndoClear()` — очищает буфер отмены изменений, чтобы пользователь не смог откатить произведенные действия.
- `UnProtect()` — снимает защиту с документа (определенную методом `Protect()` или в графическом интерфейсе). Может быть очень полезным перед программным внесением изменений в защищенный документ.

Часто используемых событий у объекта `Document` всего три: `New` (можно определить только для шаблона, срабатывает, когда на основе этого шаблона создается новый документ), `Open` и `Close`. Все эти события очевидны и изначально доступны в окне редактора кода Visual Basic.

10.5. Объекты *Selection*, *Range* и *Bookmark*

10.5.1. Работа с объектом *Selection*

После того как мы запустили приложение, нашли и активизировали нужный нам файл, следующее действие, которое выполняется чаще всего, — ввод или редактирование текста в нужном месте. Для этого используются объекты `Selection`, `Range` и `Bookmark`. Каждое из них используется в своих ситуациях и для своих задач.

Первый объект, который мы рассмотрим, — это объект `Selection`.

Обычно перед тем, как что-либо сделать в окне документа Word, пользователь либо выделяет нужный фрагмент текста, либо переставляет указатель вставки текста в нужное место. Объект `Selection` представляет именно такой выделенный участок текста (а если ничего не выделено пользователем, то место, где находится указатель вставки). Именно этот объект обычно использует макрорекордер.

Создавать объект `Selection` и получать на него ссылку в переменную совершенно не нужно. Дело в том, что объект `Selection` в документе может быть

только один. Он создается автоматически при запуске Word и всегда доступен. Обращаться к нему можно так:

```
Application.Selection.Text = "Вставляемый текст"
```

или просто:

```
Selection.Text = "Вставляемый текст"
```

Обычно нам нужно правильно определить то место, на которое указывает объект `Selection`, чтобы выделить нужный нам участок текста или точку для ввода.

Существует несколько способов для настройки выделения в документе Word:

- самый простой способ — положиться на выделение нужного текста пользователем. Обычно такой способ применяется для сложного редактирования или форматирования участков текста и для ввода информации в указанное пользователем место документа, когда в автоматическом режиме нужное место не найти;
- воспользоваться методом `Select()`, который предусмотрен для огромного числа объектов (`Document`, `Range`, `Bookmark`, `Table` со всеми подобъектами типа столбцов и строк, `PageNumber`, `Field` и т. п.). Этот метод просто выделяет весь документ, закладку, таблицу и т. п.;
- воспользоваться многочисленными методами объекта `Selection`, чтобы преобразовать уже существующее выделение;
- воспользоваться объектом `Find` для поиска нужного фрагмента текста. Подробнее об этом объекте — в *разд. 10.6.5*;
- если вам нужно ввести информацию в самое начало документа, можно вообще ничего не делать. По умолчанию указатель вставки устанавливается на начало документа. Только не забудьте сделать этот документ активным.

Если вы полагаетесь на выделение нужного места пользователем, то помните, что пользователь может ухитриться выделить одновременно несмежные участки текста (при помощи клавиши <Ctrl>) или выделить не текст, а часть таблицы, рисунок или другой нестандартный объект в документе. Чаще всего поведение программы, работающей с объектом `Selection`, в этом случае становится совершенно непредсказуемым, поэтому рекомендуется всегда использовать дополнительные проверки при помощи свойств `Type` и `Information` объекта `Selection`.

Несмотря на то, что применение объекта `Selection` — самый простой и наглядный метод редактирования текста, и чаще всего именно он используется макрорекордером, на практике программисты применяют его редко. Объясняется это очень просто: при использовании этого объекта мы слишком зави-

сим от действий пользователя. Если во время выполнения нашего кода пользователь проявит инициативу и начнет щелкать по документу мышью, результат может быть непредсказуемым. Защититься от вмешательства пользователя можно двумя способами:

- работать со скрытым (т. е. невидимым) документом или, возможно, со скрытым экземпляром Word. Для включения и отключения видимости можно использовать свойство `Visible` объектов `Document` или `Application`;
- более удобный способ — вместо объекта `Selection` использовать объекты `Range` и `Bookmark`, о которых будет рассказано в следующих разделах.

10.5.2. Свойства и методы объекта *Selection*

Вначале расскажем о самых часто используемых свойствах объекта `Selection`.

- `Bookmarks` — возвращает коллекцию `Bookmarks`, т. е. все закладки, которые имеются в выделенном фрагменте текста. Закладки — один из самых часто используемых объектов в приложениях VBA с использованием Word. Подробнее о них будет рассказано в *разд. 10.5*.
- `Start` и `End` — свойства, которые определяют номера первого и последнего символа в выделении (по отношению к тексту документа или другим его частям, например, к сноске). Первая позиция в тексте документа — всегда 0. Если вы создаете документ из неизменяемого шаблона, можно использовать эти свойства, чтобы найти нужное место в документе для ввода текста (однако этот способ не очень рекомендуется, потому что при правке шаблона вам придется править много программного кода).
- `ExtendMode` — переключает пользователя в режим выделения текста, когда нажатие клавиш со стрелками, <Home> и <End> приводит не к перемещению указателя ввода, а к изменению выделения.
- `Find` — очень важное свойство, которое возвращает объект `Find`. Подробнее об этом объекте и о его вложенном объекте `Replace` будет рассказано в *разд. 10.6.5*.
- `Flags` — свойство, которое позволяет проверить или изменить некоторые моменты, связанные с выделением: является ли оно активным, находится ли в конце строки и т. п. Регулирует одновременно пять параметров при помощи битовой маски.
- `Font` — возвращает объект `Font`, при помощи которого можно управлять оформлением текста в выделении. Доступны все возможности, которые есть на графическом интерфейсе в меню **Формат | Шрифт**. Например, чтобы назначить выделенному тексту шрифт Arial 10 pt, можно использовать код:

```
Selection.Font.Name = "Arial"
Selection.Font.Size = 10
```

- `Information` — важнейшее свойство объекта `Selection` для целей проверок. Возвращает огромное количество информации о выделении (в какой части документа, внутри таблицы или нет, включены ли клавиши <CapsLock> и <NumLock>, включен ли режим "Замена" при вводе текста, на какой странице находится выделение и сколько страниц и т. п.).
- `IPAtEndOfLine` — возвращает `True`, если курсор ввода текста (*insertion point* — IP) находится в конце строки (в крайнем правом положении при выравнивании).
- `LanguageId` — позволяет пометить выделение, как написанное на определенном языке. Правильное определение языка позволяет избежать проблем при проверке орфографии.
- `NoProofing` — отменяет для выделения проверку орфографии и грамматики. Очень рекомендуется помечать таким образом текст с программным кодом, списками фамилий, названиями фирм, специфическими терминами и т. п.
- `Range` — создает из выделения объект `Range`.
- `StoryType` — еще одно свойство для проверок. Определяет тип текста документа, в котором находится выделение.
- `Text` — самое важное свойство объекта `Selection`. Позволяет ввести текст на месте выделения (или в том месте, где стоит указатель). Например, чтобы 100 раз напечатать текст "Привет!", можно воспользоваться кодом:

```
For i = 0 To 100
    Selection.Text = "Привет!"
    Selection.EndOf
Next
```

  Метод `EndOf()` здесь позволяет перейти в конец текущего выделения. Он нужен здесь для того, чтобы не перезаписывать один и тот же текст 100 раз, поскольку после ввода текст остается выделенным.
- `Type` — еще одно проверочное свойство, которое позволяет предупредить ошибку, если пользователь выделил что-то неположенное. Например, при обычном выделении значение этого свойства будет равно 1, а если выделены несмежные участки текста — 2.
- `Words` — позволяет вернуть коллекцию `Words`. Эта коллекция состоит из набора объектов `Range`, каждому из которых соответствует слово в выделенном тексте.

Методов у объекта `Selection` гораздо больше, чем свойств.

- `Calculate()` — позволяет посчитать математическое выражение прямо в процессе ввода текста и вернуть его результат (используя только тип данных `Single`).
- `ClearFormatting()` — очищает форматирование (и на уровне текста, и на уровне параграфа). Этот метод можно применять не только для объекта `Selection`, но и для объектов `Find` и `Replace`.
- `Collapse()` — превращает выделение в указатель вставки. Можно использовать два варианта: указатель вставки помещается на начало выделения или на конец выделения. Очень удобно, если вам требуется только вставить новый текст без удаления старого.
- `Copy()`, `CopyAsPicture()`, `Cut()`, `Paste()` и `Delete()` — эти методы можно использовать для копирования выделенного участка документа, копирования и вставки в виде изображения, вырезания, вставки и удаления соответственно.
- `EndKey()` — этот метод так называется, поскольку он очень похож по функциональности на нажатие клавиши <End>. Он позволяет (в зависимости от переданных параметров) перейти на конец строки, столбца или записи в таблице (по умолчанию на конец строки) и либо выделить до этого места, либо установить на нем указатель вставки. Чтобы перевести курсор вставки на конец текста документа, можно воспользоваться кодом:

  ```
  Selection.EndKey Unit:=wdStory, Extend:=wdMove
  ```

  Если же нужно перейти на начало элемента, используется аналогичный метод `HomeKey()`.
- `EndOf()` — по функциональности практически идентичен методу `EndKey()`. Он позволяет перейти на конец символа, слова, предложения, абзаца, секции, текста документа, таблицы и т. п. Различие между этими методами заключается в том, что `EndKey()` работает только с текущим элементом текста, а при помощи `EndOf()` можно, например, найти следующую таблицу в выделенной части документа и перейти на ее конец. Чтобы перейти на начало элемента текста, существует метод `StartOf()`.
- `Expand()` — расширяет выделение на слово, предложение, абзац и т. п., в зависимости от переданного параметра. Метод `Extend()` позволяет расширить выделение (вместо слова — предложение, вместо предложения — абзац и т. п.). Метод, обратный методу `Expand()`, — `Shrink()`.
- `GoTo()` — работает практически аналогично такому же методу объекта `Document`.

- `GotoNext()` — перейти на следующую строку, страницу, закладку и т. п. Аналогично работает метод `GotoPrevious()` (переход на предыдущий элемент).
- Назначение многочисленных методов с префиксом `Insert...` очевидно. Чаще всего используются методы `InsertBefore()` (вставить перед выделением) и `InsertAfter()` (вставить после выделения).
- Методы с префиксом `Move...` также встречаются едва ли не в любой программе, связанной с вводом текста в Word. Самые важные и удобные из этих методов:
  - `MoveLeft()`, `MoveRight()`, `MoveUp()`, `MoveDown()`, `MoveEnd()`, `MoveStart()` — эти методы позволяют переместить выделение соответственно влево, вправо, вверх, вниз, к концу существующего выделения или к его началу. Каждый из этих методов принимает дополнительные параметры, при помощи которых можно определить, на сколько символов будет перемещаться указатель, будет ли двигаться выделение, распространяясь на новую область, и т. п.;
  - `MoveStartUntil()`, `MoveStartWhile()`, `MoveEndUntil()`, `MoveStartWhile()` — отличаются тем, что курсор вставки перемещается не на определенное количество символов, а пока не будет найдено (или пока встречается) определенная последовательность символов. Также очень удобно использовать эти методы для установки курсора в нужное место в документе для ввода текста;
  - `Move()` — более гибкий метод. Он позволяет отсчитывать не только определенное количество символов, но и слов, предложений, абзацев, разделов, столбцов и строк в таблице и т. п. Позволяет обойтись минимальным количеством изменений в коде, если изменился исходный шаблон для ввода данных.
- `Next()` — позволяет перейти вперед на определенное количество символов, слов, предложений, абзацев, разделов, столбцов и строк в таблице и т. п. Переход назад осуществляет метод `Previous()`.
- `NextField()` — позволяет перейти на следующее поле в форме или проверить, не кончились ли поля (в этом случае метод вместо объекта `Field` возвратит `Nothing`). Есть также метод `PreviousField()`.
- `SelectColumn()`, `SelectRow()`, `SelectCell()` — очень удобные методы для выполнения различных операций в таблице Word.
- `SelectCurrentAlignment()`, `SelectCurrentFont()`, `SelectCurrentIndent()`, `SelectCurrentColor()` и т. п. — выделяют текст до изменения выравнивания, шрифта, отступа, цвета и т. п. Также очень удобно использовать эти

методы для форматирования или для выделения специальным образом добавленного текста.

- `SetRange()` — самый простой способ настроить выделение. Передаются номера первого и последнего символов того фрагмента текста, который нужно выделить. Нумерация начинается с 0, скрытые служебные символы также считаются. Такой же метод существует у объекта `Range`.
- `Sort()`, `SortAscending()`, `SortDescending()` — позволяют отсортировать по алфавиту, датам и т. п. абзацы или столбцы в таблице (которые входят в выделение). Этот метод поможет сэкономить вам массу времени и сил.
- `ToggleCharacterCode()` — позволяет ввести код служебного символа и тут же преобразовать его в символ Unicode. Например, чтобы ввести символ Евро, можно воспользоваться командами:

```
Selection.TypeText Text:="20ac"
Selection.ToggleCharacterCode
```

- `TypeText()` — самый простой, надежный и часто используемый метод ввода текста. Принимает единственный параметр — текст, который нужно ввести. Будет ли перезаписан текущий текст выделения, зависит от свойства `ReplaceSelection` объекта `Options` *(см. разд. 10.6.4)*.
- `WholeStory()` — выделяет текущую часть документа. Обычно используется, чтобы выделить текст документа без сносок, редакторской правки, колонтитулов и т. п.

10.5.3. Работа с объектом *Range*, его свойства и методы

Как уже говорилось, чаще всего разработчиками для определения места ввода текста и навигации по документу используется объект `Selection`. Для этих же целей можно использовать и объект `Range`. Главное отличие между этими объектами заключается в том, что объект `Selection` может определить сам пользователь (выделив текст мышью), а объект `Range` можно определить только программно, независимо от текущего положения указателя и действий пользователя.

Рекомендуется всегда использовать объект `Range` вместо объекта `Selection`. Тем самым вы защитите себя от возможных ошибок, связанных с действиями пользователя (например, если пользователь в момент, когда программно вводится текст, щелкнет мышью по какому-либо месту в документе).

Формальное определение объекта `Range` выглядит так: это программный объект, который представляет непрерывный фрагмент текста в документе. Этот

объект не зависит от объекта Selection — вы можете работать с объектом Range, не меняя текущего выделения. Он может не включать в себя ни одного символа (представлять собой указатель ввода текста).

Объектов Range в каждый момент времени может быть сколько угодно, а объектов Selection — только один.

Объект Range можно создать несколькими способами:

- первый способ — воспользоваться методом Range() объекта Document. В этом случае вам потребуется передать номера начального и конечного символов диапазона, а также текст документа, в который будут отсчитываться эти символы. Например, создать диапазон, который будет включать в себя первые 10 символов документа, можно так:

```
Dim rngDoc As Range
Set rngDoc = ActiveDocument.Range(Start:=0, End:=10)
```

- второй способ — воспользоваться свойством Range, которое предусмотрено для большого количества объектов (Bookmark, Selection, Table-Row-Cell, Paragraph и т. п.). В этом случае при помощи этого свойства мы получаем объект Range, представляющий данный объект;
- третий способ — воспользоваться большим количеством вспомогательных свойств (Characters, Words, Sentences и т. п.), которые делят текст на отрезки — объекты Range. Эти свойства возвращают коллекции объектов Range. Конечно, если вы создаете коллекцию объектов Range, представляющих каждый символ большого документа, то с точки зрения производительности такое решение может быть не самым лучшим;
- четвертый способ — переопределить существующий объект Range. Обычно для этой цели используется метод SetRange() объекта Range;
- и, наконец, пятый способ — самый удобный в реальных приложениях. Он заключается в том, что вы вначале создаете шаблон нужного вам документа (договора, приходного ордера, отчета и т. п.), в который при создании помещаете закладки в те места, где потом потребуется вставить данные. Затем программным способом для каждой закладки создается объект Range, и уже с его помощью производится ввод информации (данные о заказчике, сумма в кассовом ордере и т. п.).

Для целей отладки (чтобы убедиться, что объект Range действительно включает в себя тот фрагмент текста, который вы планировали) можно создавать на основе объекта Range объект Selection (т. е. выделять диапазон). Для этого у объекта Range предусмотрен метод Select().

Большая часть свойств и методов объекта Range совпадает с аналогичными свойствами и методами объекта Selection (который мы уже рассмотрели),

поэтому приводить здесь мы их не будем. Точно так же для объекта `Range` чаще всего используется одно-единственное свойство `Text`, которое позволяет вставить в место в документе, представленное этим объектом, нужный текст. Далее приведена информация о некоторых уникальных методах объекта `Range`.

- `InsertDatabase()` — возможно это самый простой метод вставить результат запроса к базе данных в файл Word. Генерирует таблицу на основе возвращаемого результата запроса и вставляет ее в место, определенное объектом `Range`. Умеет работать по ODBC, DDE, есть встроенные средства работы с Access. Возможности этого метода очень ограничены, поэтому рекомендуется использовать его только в самых простых случаях. Во всех остальных случаях лучше использовать объектную библиотеку ADO *(см. гл. 9)*. Пример обращения к файлу Access средствами этого метода может выглядеть так:

```
With Selection
    .Collapse Direction:=wdCollapseEnd
    .Range.InsertDatabase _
        Format:=wdTableFormatSimple2, Style:=191, _
        LinkToSource:=False, _
        Connection:="Table.Поставщики", _
        DataSource:="C:\Program Files\Microsoft
Office\OFFICE11\SAMPLES\Борей.mdb"
End With
```

- `IsEqual()` — сравнивает два объекта `Range` (или объект `Selection` и объект `Range`). Если совпадают начальная позиция в документе, конечная позиция и текст фрагмента, возвращается `True`.
- `FoneticGuide()` — позволяет вставить транскрипцию над текстом в документе.
- `Relocate()` — переставляет местами абзацы в диапазоне.
- `Select()` — как уже говорилось ранее, создает объект `Selection` на основе объекта `Range` (т. е. просто выделяет весь текст в этом объекте). Очень удобно использовать для отладочных целей.
- `SetRange()` — один из самых важных методов объекта `Range`. Позволяет изменять этот объект. Например, получить объект `Range`, в который будет входить текст от начала документа до текущей позиции курсора (или до конца выделения), можно при помощи команд:

```
Dim MyRange As Range
Set MyRange = ActiveDocument.Range(Start:=0, End:=0)
MyRange.SetRange Start:=MyRange.Start, End:=Selection.End
MyRange.Select    'для демонстрации
```

10.5.4. Объект *Bookmark*

Объект `Bookmark` — это просто закладка. На практике это самый удобный способ навигации по документам, созданных при помощи шаблонов (например, отчетов). Принципиальное отличие его от объектов `Selection` и `Range` заключается в том, что все выделения и диапазоны теряются при закрытии документа (объекты `Range` вообще существуют только во время работы создавшей их процедуры, а закладки сохраняются вместе с документом. Если документ создан на основе шаблона, то все закладки, которые были определены в этом шаблоне, будут определены и в созданном на его основе документе.

Создать закладку (с помощью меню **Вставка | Закладка**) намного проще, чем считать количество символов для объекта `Range` от начала документа, абзаца или предложения или выполнять операции `Move()` (`MoveDown()`, `MoveRight()`, `MoveNext()`) для объекта `Selection`. Кроме того, если вы будете исправлять шаблон (а делать это приходится очень часто), вам, скорее всего, не придется править код для определения места вставки (что потребуется для объектов `Selection` и `Range`).

Функциональность объекта `Bookmark` невелика. Свойств и методов у этого объекта намного меньше, чем у объектов `Selection` и `Range`. Однако обычно никто и пытается использовать объект `Bookmark` для работы с текстом. Из объекта `Bookmark` (все закладки собраны в коллекцию `Bookmarks`, принадлежащей документу) очень просто получить объект `Selection` (при помощи метода `Select()`) или объект `Range` (при помощи свойства `Range`), и дальше можно пользоваться уже свойствами и методами этих объектов, например:

```
ThisDocument.Bookmarks("Bookmark1").Select
MsgBox Selection.Text
```

Создавать объекты `Bookmark` программным способом необязательно, но если есть необходимость, то можно использовать метод `Add()` коллекции `Bookmarks`:

```
ThisDocument.Bookmarks.Add Name:="temp", Range:=Selection.Range
```

У этого метода всего лишь два параметра, которые и используются в примере.

Некоторые важные свойства объекта `Bookmark` представлены далее.

- `Empty` — если это свойство возвращает `True`, то закладка указывает на указатель вставки, а не на текст;
- `Name` — имя закладки. Очень удобно, что найти нужную закладку в коллекции `Bookmarks` можно не только при помощи индекса (номера) закладки, но и по ее имени.

- `Range` — возвращает объект `Range` на месте этой закладки.
- `Start`, `End`, `StoryType` — эти свойства аналогичны таким же у объекта `Selection`.

Методов у объекта `Bookmark` всего три.

- `Copy()` — создает закладку на основе существующей.
- `Delete()` — удаляет закладку.
- `Select()` — выделяет то, на что ссылается закладка.

10.6. Другие объекты Word

В объектной модели Word сотни объектов, и подробно о каждом из них в рамках этой книги рассказать невозможно. Скорее всего, большая часть из них вам никогда не потребуется, а другие можно будет легко найти при помощи макрорекордера. Далее приведена обзорная информация по самым важным из еще не рассмотренных объектов Word. Многие эти объекты доступны через одноименные свойства объекта `Application` и уже упоминались при рассмотрении соответствующих свойств.

10.6.1. Коллекция *AddIns* и объекты *AddIn*

Коллекция `AddIns` состоит из объектов `AddIn`, которые представляют собой шаблоны Word и приложения, встраиваемые в Word. Важная возможность этой коллекции заключается в том, что при помощи метода `Add()` можно в автоматическом режиме устанавливать шаблоны и надстройки (в графическом режиме это можно сделать через меню **Сервис | Шаблоны и надстройки**). Если вы активно используете эти средства в своих приложениях, то имеет смысл подумать над реализацией проверки наличия нужного шаблона или надстройки.

Шаблоны — это файлы с расширением dot, которые служат образцами для создания документов Word. Чаще всего они используются для того, чтобы защитить от пользователя сохраненные начальные "заготовки" отчетов, или как хранилища стилей, макросов, параметров и т. п. для сложных документов, для которых требуется стандартизация (например, в издательствах для рукописей и оригинал-макетов).

Надстройки — это откомпилированные модули DLL (для них используется расширение wll — *Word Add-In Library*). Это специальные приложения, которые изначально предназначены для встраивания в Word. Поскольку они откомпилированы и могут быть написаны на любом COM-совместимом языке, например, C++, Visual Basic или Delphi, то они работают намного быстрее,

чем родные программы VBA — макросы. Поэтому есть смысл задуматься об использовании надстроек, если вам нужно серьезное увеличение производительности.

10.6.2. Объект *AutoCorrect*

При помощи этого объекта настраиваются те параметры, которые можно на графическом экране найти в меню **Сервис | Параметры автозамены**. Эти настройки влияют на автоматическое исправление текста, вручную вводимого пользователем. Можно использовать для единообразного написания названия фирмы и прочих элементов, которые у пользователей часто получаются разными, но чаще всего этот объект используется для отключения автозамены, поскольку это может мешать пользователям. Например, чтобы очистить весь список **Заменять при вводе** на первой вкладке параметров автозамены, можно использовать код:

```
For Each Item In AutoCorrect.Entries
    Item.Delete
Next
```

10.6.3. Коллекция *Languages* и объект *Language*

Коллекция `Languages` представляет языки, которые знает данная версия Word и с которыми умеет работать. Чтобы просмотреть текущий набор языков, с которыми умеет работать Word, можно воспользоваться кодом:

```
For Each lan In Languages
    Debug.Print lan.Name
    i = i + 1
Next lan
Debug.Print i
```

10.6.4. Объект *Options*

Этот объект с огромным количеством свойств позволяет настраивать практически любые свойства самого приложения Word и текущего документа. В нем есть все, что доступно через вкладку **Сервис | Настройка** и еще множество параметров. Дочерних объектов у `Options` нет — все, что в нем есть, доступно через его свойства. Например, чтобы включить в Word отображение белых букв на синем фоне (мой привычный режим работы), можно использовать команду:

```
Options.BlueScreen = True
```

10.6.5. Объекты *Find* и *Replacement*

Объекты `Find` и `Replacement`, как понятно из их названий, предназначены для выполнения операций поиска и замены. Объект `Find` — это условия поиска, "упакованные" в программный объект. У него множество свойств (`Text`, `Style`, `Font`, `Forward`, `MathCase`, `LanguageID` и т. п.), которые позволяют определить эти условия. Чтобы запустить поиск, используется метод `Execute()` со множеством необязательных параметров, которые во многом дублируют свойства этого объекта, а передав параметр `ReplaceWith`, можно выполнить даже замену текста. Для того чтобы заменить определенное слово во всем документе или просто проверить результаты поиска, используется значение, возвращаемое методом `Execute()`. Если поиск был успешен, то возвращается `True` (и, возможно, нужно будет продолжить поиск в оставшейся части документа), а если нет — `False`.

Как будет работать объект `Find` зависит от того, из какого объекта он был создан. Если он был создан при помощи свойства `Find` объекта `Selection`, то при обнаружении нужный фрагмент выделяется. Если он был создан при помощи такого же свойства объекта `Range`, то диапазон переопределяется на найденный текст. Например, чтобы найти и выделить следующее вхождение текста `"2004"`, можно использовать код:

```
Selection.Find.Text = "2004"
Selection.Find.Execute
```

Объект `Replacement` точно так же хранит настройки замены. К этому объекту можно обратиться при помощи свойства `Replacement` объекта `Find`. Например, чтобы заменить везде до конца документа текст `"2004"` на `"2005"`, можно использовать код:

```
Selection.Find.Text = "2004"
Selection.Find.Replacement.Text = "2005"
Selection.Find.Execute Replace:=wdReplaceAll
```

Важный момент: объекты `Find` и `Replacement` по умолчанию производят поиск и замену с учетом форматирования. Поэтому если одно и то же слово будет оформлено в документе разным шрифтом, оно будет считаться разными словами. Поэтому многие программисты используют метод `ClearFormatting()`. Этот метод очищает форматирование внутри объектов `Find` и `Replacement` (на сам документ это никак не влияет) и позволяет производить поиск без учета форматирования. Например, наш код при использовании этого метода может выглядеть так:

```
Selection.Find.Text = "2004"
Selection.Find.ClearFormatting
```

```
Selection.Find.Replacement.Text = "2005"
Selection.Find.Replacement.ClearFormatting
Selection.Find.Execute Replace:=wdReplaceAll
```

10.6.6. Объекты *Font* и *ParagraphFormat*

Объекты `Font` и `ParagraphFormat` ответственны за форматирование соответственно участков текста и абзацев. Свойства объекта `Font` позволяют определить все параметры, которые доступны через пункт меню **Формат | Шрифт**, а свойства объекта `ParagraphFormat` — через **Формат | Абзац**. Объект `Font` можно получить через свойство `Font`, которое есть, в частности, у объектов `Selection`, `Range` и `Find`, а объект `ParagraphFormat` — через свойство `Format`, которое есть у объектов `Paragraph` (для одного абзаца) и коллекции `Paragraphs` (для нескольких абзацев). Свойство `Format`, которое возвращает объект `ParagraphFormat`, есть и у объекта `Find`.

Свойств у объектов `Font` и `ParagraphFormat` множество, но все они очевидны. Например, чтобы назначить выделенному тексту шрифт Arial и сделать его полужирным, можно использовать код:

```
Selection.Font.Name = "Arial"
Selection.Font.Bold = True
```

а чтобы назначить первому абзацу текста выравнивание по ширине, можно использовать команду:

```
Paragraphs(1).Format.Alignment = wdAlignParagraphJustify
```

10.6.7. Объект *PageSetup*

Если в вашем приложении используется печать, то чаще всего без использования объекта `PageSetup` не обойтись. Он позволяет программным образом настроить то, что на графическом экране настраивается через меню **Файл | Параметры страницы**. Объект `PageSetup` является вложенным в объекты `Document`, `Selection` и `Range`, и обычно обращение к нему происходит через эти объекты. Например, чтобы при печати документ был выведен в альбомной ориентации, можно воспользоваться командой:

```
ThisDocument.PageSetup.Orientation = wdOrientLandscape
```

10.6.8. Объекты *Table*, *Column*, *Row* и *Cell*

Если в вашем приложении нужно вывести какой-то стандартный документ по утвержденной форме (платежка, кассовый ордер, командировочный отчет и т. п.), то, скорее всего, для форматирования вы будете использовать табли-

цу. Таблица во многих ситуациях позволяет гарантировать правильное расположение данных в форме относительно друг друга. Многие программисты изначально используют таблицу даже для документов, которые на таблицы похожи мало (сетку таблицы всегда можно скрыть, а ячейки — слить между собой).

Создание таблицы начинается с того, что в коллекцию `Tables` (она предусмотрена для объектов `Document`, `Selection` и `Range`) добавляется новый объект `Table` (в данном примере с тремя строками и четырьмя столбцами):

```
Set Range1 = ThisDocument.Range(Start:=0, End:=0)
Dim Table1 As Table
Set Table1 = ThisDocument.Tables.Add(Range1, 3, 4)
```

Затем можно настроить свойства таблицы, например, воспользовавшись методом `AutoFormat()` (возможности у него те же, что доступны через меню **Таблица | Автоформат**):

```
Table1.AutoFormat wdTableFormatGrid5
```

Чаще всего нам нужно ввести какие-либо данные в ячейку таблицы. Ячейку таблицы в Word представляет объект `Cell`. Мы можем добраться до нужной ячейки через объекты `Columns` и `Rows`, `Selection` и `Range`, однако удобнее всего сделать так:

```
Table1.Cell(1, 1).Range.InsertAfter "10"
Table1.Cell(2, 1).Range.InsertAfter "15"
Table1.Cell(3, 1).AutoSum
```

Мы ввели в первую строку первого столбца значение 10, во вторую строку первого столбца — значение 15, а в третьей строке мы просуммировали значения по всему столбцу. Таблицы Word — это, конечно, не Excel, но при помощи метода `Formula()` объекта `Cell` в таблицу можно вставлять достаточно сложные вычисляемые значения.

Если вы используете Word для самой распространенной цели — вывод данных, полученных из другого приложения или базы данных, то, вполне возможно, заниматься программированием таблиц вам вообще будет не нужно. Достаточно будет создать и оформить таблицу в шаблоне на графическом экране и пометить места ввода данных при помощи закладок.

10.6.9. Объект *System*

При помощи объекта `System` можно получить большое количество информации о системе, в которой работает ваше приложение. В качестве альтернативы этому объекту можно подумать об использовании объектных моделей

Windows Script Host и WMI — возможностей для работы с системой в этих объектных моделях намного больше. Однако если у вас в организации используются старые операционные системы (Windows NT, Windows 98), где они могут быть не установлены, использование этого объекта очень удобно.

Далее представлены главные свойства объекта `System`.

- `CountryRegion` — возвращает текущие региональные настройки операционной системы. Если на вашем компьютере установлены русские региональные настройки, то возвращается значение 7 (несмотря на то, что его нет в документации), если установлены американские — 1.
- `FreeDisk` — возвращает объем доступного дискового пространства для пользователя на текущем диске (можно сменить текущий диск). Если документы очень большие и иногда возникают проблемы с местом на диске, можно реализовать проверку наличия свободного места, например, при выполнении операций сохранения.
- `HorizontalRezolution` и `VerticalResolution` — возможность получить информацию о текущем разрешении экрана пользователя, например, для правильного отображения больших форм.
- `LanguageDesignation` — определяет язык интерфейса операционной системы. Возвращается в виде строкового значения, например:

  ```
  Debug.Print System.LanguageDesignation
  ```

- `OperationSystem` — возвращает информацию об операционной системе. К сожалению, предусмотрено только два значения: `"Windows"` — для линейки Windows 95/98/ME и `"Windows NT"` — для линейки NT/2000/XP/2003.

У этого объекта есть всего два метода: `Connect()` (подключить сетевой диск) и `MsInfo()` (показать окно системной информации).

10.6.10. Коллекция *Tasks* и объект *Task*

Чаще всего Word запускается из Excel, Access или другого приложения, но иногда встречается и обратная необходимость — нужно открыть из Word другое приложение и переключиться в него. Самый простой способ запустить другое приложение из Word — воспользоваться стандартным объектом VBA `Shell`. Например, чтобы запустить блокнот, можно воспользоваться командой:

```
Shell "notepad.exe"
```

Есть и множество других возможностей, например, воспользоваться объектом `Application` для других приложений Word, или средствами Windows Script Host (особенно для консольных приложений), или средствами WMI, если приложение нужно запустить на другом компьютере.

После того, как приложение запущено, весь набор работающих приложений представляется в Word коллекцией `Tasks`, а каждое отдельное приложение — объектом `Task`. У коллекции `Tasks` есть два интересных метода.

- `Exists()` — проверяет, запущено ли нужное нам приложение. Например, запуск нашего блокнота с проверкой может выглядеть так:

```
If Tasks.Exists("Notepad") = False Then
    Shell "notepad.exe"
Else
    Tasks("Notepad").Activate
End If
Tasks("Notepad").WindowState = wdWindowStateMaximize
```

- `ExitWindows()` — производит операцию Log Off, т. е. завершает сеанс работы в Windows. Несохраненные документы Word при этом закроются без сохранения (и без вопросов к пользователю), а документы остальных приложений пользователю будет предложено сохранить.

У объекта `Task` интересных свойств и методов несколько больше.

- `Height`, `Width`, `Top`, `Left` — эти свойства позволяют точно настроить размер окна выбранного вами приложения.
- `Visible` — позволяет спрятать приложение.
- `WindowState` — позволяет развернуть, свернуть или восстановить окно приложения.
- `Activate()`, `Close()`, `Move()`, `Resize()` — назначение этих методов очевидно.
- `SendWindowMessage()` — самый интересный метод. Он позволяет передавать окну приложения сообщения Windows (щелчки мышью, нажатия клавиш и т. п.). Разобраться в том, какие сообщения можно посылать окнам приложений и что они означают, можно при помощи дополнительной документации из набора Microsoft Platform Software Development Kit (его можно скачать с сайта Microsoft). Например, чтобы в нашем блокноте отобразить окно **О программе**, можно воспользоваться командой:

```
Tasks("Notepad").SendWindowMessage &H111, 11, 0
```

10.6.11. Коллекция *Windows* и объект *Window*

Коллекция `Windows` и объект `Window` представляют собой окна открытых документов Word и используются в основном для настройки внешнего вида этих окон и навигации по ним. Получить доступ к окну нужного нам документа можно так:

```
Dim Window1 As Window
Set Window1 = Windows("doc2.doc")
```

или так:

```
Set Window1 = ThisDocument.ActiveWindow
```

После этого можно использовать свойства и методы объекта `Window`. Все они очень просты. Например, чтобы поменять заголовок окна, можно использовать такой код:

```
Window1.Caption = "Мое приложение"
```

Задание для самостоятельной работы 10: Программное формирование документа в Word

Ситуация:

Вам необходимо автоматизировать формирование договоров в виде документов Word. Типичный договор выглядит так, как представлено на рис. 10.2 (для простоты в этом задании вам нужно будет формировать только его начало). Изменяемые данные, которые должны подставляться программно, выделены зеленым цветом.

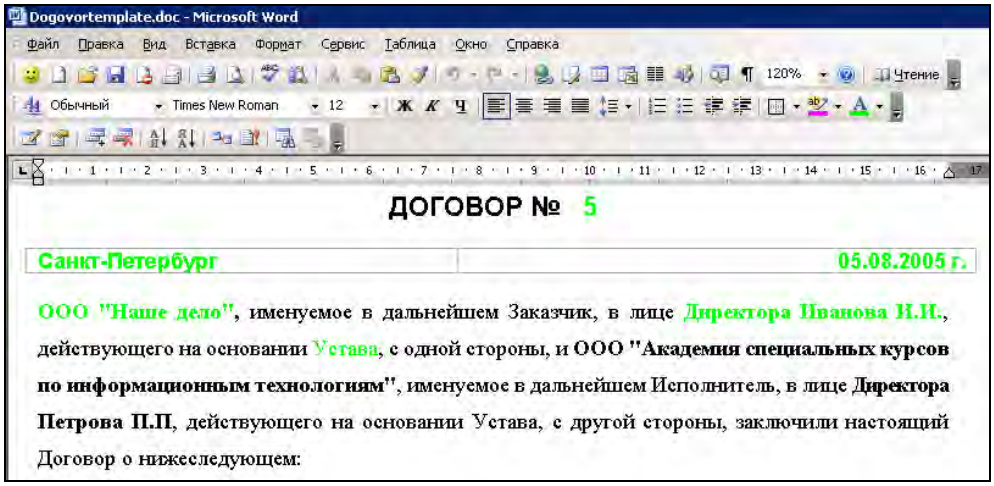


Рис. 10.2. Так должен выглядеть созданный программно договор

ЗАДАНИЕ:

1. Создайте в шаблоне Normal.dot пользовательскую форму с именем `FormDog` и заголовком **Данные договора**, аналогичную представленной на рис. 10.3.

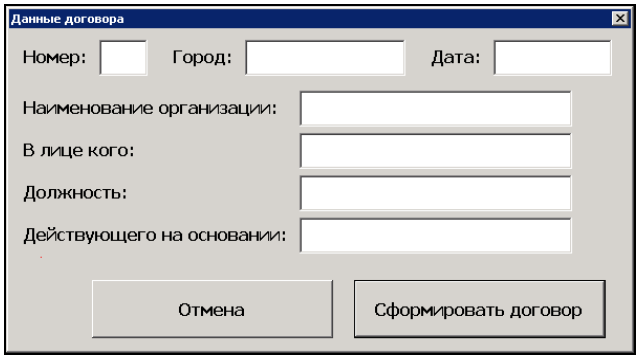


Рис. 10.3. Форма для занесения данных договора

2. Создайте макрос, по которому должна открываться эта форма, и назначьте этому макросу кнопку на панели инструментов Word.
3. Создайте и сохраните шаблон на диске с именем C:\DogovorTemplate.dot, в который будут подставляться необходимые данные, и добавьте в нужные места закладки.
4. Создайте для кнопки **Сформировать договор** на форме программный код, при помощи которого на основе шаблона и подставляемых данных из формы формировался бы новый документ с текстом договора.

Примечание

В реальном приложении данные из формы должны были бы сохраняться в базе данных и использоваться потом многократно (например, для формирования других документов). В этом задании для простоты данные для создаваемого документа берутся напрямую из формы. Работа с базой данных в похожей ситуации будет рассмотрена в задании к *гл. 12*. По этой же причине все данные в этом примере (включая дату и номер договора) текстовые.

Ответ к заданию 10

К пункту 1 задания (создание пользовательской формы):

1. Откройте окно редактора Visual Basic для Word и щелкните правой кнопкой мыши по проекту **Normal** в **Project Explorer**, а затем выберите в контекстном меню **Insert | UserForm**.
2. В дизайнере форм сконструируйте форму, аналогичную представленной на рис. 10.3 (про работу с дизайнером форм рассказывалось в *гл. 5*). Пусть в нашем примере элементы управления на форме называются так:
   - `txtNumber` — текстовое поле для ввода номера договора;
   - `txtCity` — текстовое поле для ввода города;

- txtDate — текстовое поле для ввода даты;
- txtOrg — текстовое поле для ввода наименования организации;
- txtPerson — текстовое поле для ввода представителя организации;
- txtTitle — текстовое поле для ввода его должности;
- txtLaw — текстовое поле для ввода юридического основания;
- cmdDog — кнопка для формирования договора;
- cmdCancel — кнопка **Отмена**.

3. Настройте оформление для элементов управления по вашему вкусу. Установите значение для свойства Caption для формы как "Данные договора". Для кнопки cmdDog установите True для значения свойства Default, а для кнопки cmdCancel установите True для значения свойства Cancel. Для свойства Name самой формы введите значение FormDog.

К пункту 2 задания (создание макроса и кнопки для отображения формы):

1. В стандартном модуле NewMacros проекта **Normal** создайте новую процедуру с именем FormDog(). Код ее может быть таким:

```
Public Sub FormDogShow()
    FormDog.Show
End Sub
```

   Убедитесь, что при его запуске открывается созданная вами форма.

2. В Word в меню **Сервис** выберите **Настройка**, а затем перейдите на вкладку **Команды**. В списке **Категории** выберите **Макросы**, а затем перетащите на любую панель инструментов макрос **Normal.NewMacros.FormDogShow**. Настройте для созданной кнопки подходящий формат отображения *(см. гл. 1)*. После этого закройте окно **Настройка** и убедитесь, что при нажатии на эту кнопку открывается форма.

К пункту 3 задания (создание шаблона документа Word):

1. Создайте новый документ Word, аналогичный представленному на рис. 10.4.
2. Поместите в нужные места этого документа закладки. Места вставки закладок можно посмотреть на рис. 10.2 (текст, выделенный зеленым). Пусть закладки называются так:
   - bNumber — закладка для ввода номера договора;
   - bCity — закладка для ввода города;

- `bDate` — закладка для ввода даты;
- `bOrg` — закладка для ввода наименования организации;
- `bPerson` — закладка для ввода представителя организации;
- `bTitle` — закладка для ввода его должности;
- `bLaw` — закладка для ввода юридического основания.

3. Сохраните этот файл как шаблон Microsoft Word с именем C:\DogovorTemplate.dot.

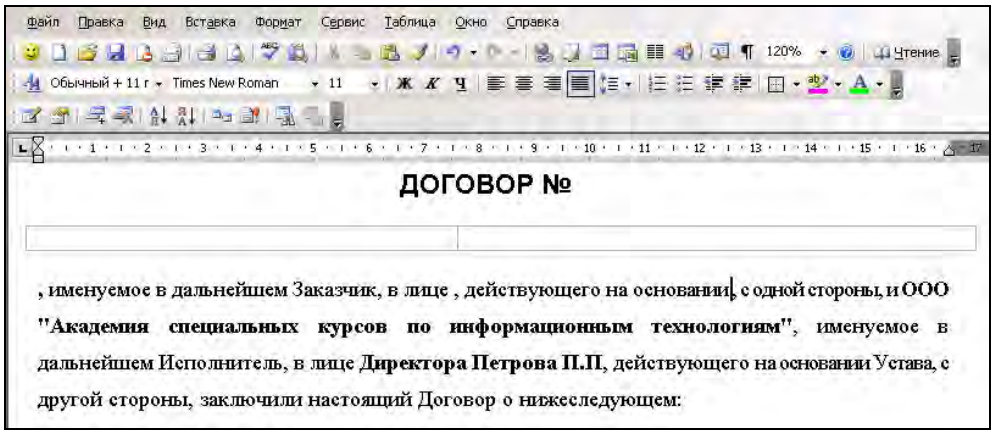


Рис. 10.4. Шаблон договора

К пункту 4 задания (создание программного кода для кнопок на форме):

1. Для события `Click` кнопки `cmdCancel` введите следующий программный код:

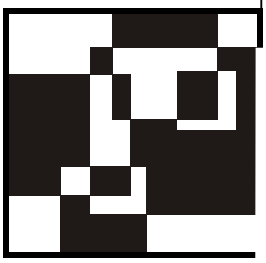
```
Private Sub cmdCancel_Click()
    FormDog.Hide
End Sub
```

2. Для события `Click` кнопки `cmdDog` можно использовать следующий программный код:

```
Private Sub cmdDog_Click()
    Dim oDoc As Document
    Set oDoc = Application.Documents.Add("C:\DogovorTemplate.dot")
    oDoc.Bookmarks("bNumber").Range.Text = txtNumber.Value
    oDoc.Bookmarks("bCity").Range.Text = txtCity.Value
    oDoc.Bookmarks("bDate").Range.Text = txtDate.Value
```

```
    oDoc.Bookmarks("bOrg").Range.Text = txtOrg.Value
    oDoc.Bookmarks("bPerson").Range.Text = txtPerson.Value
    oDoc.Bookmarks("bTitle").Range.Text = txtTitle.Value
    oDoc.Bookmarks("bLaw").Range.Text = txtLaw.Value
    FormDog.Hide
    oDoc.Activate
End Sub
```

ГЛАВА 11



Программирование в Excel

11.1. Зачем программировать в Excel

Excel — это наиболее часто используемое с точки зрения программирования приложение Office. По моему опыту преподавания курсов по программированию в Office, в подавляющем большинстве случаев сотрудников предприятий интересует, как автоматизировать выполнение операций именно в Excel. Чаще всего на предприятиях встречаются следующие ситуации:

- необходимо автоматизировать загрузку данных в таблицу Excel из базы данных, а затем в автоматическом режиме произвести обработку этой таблицы (расчеты, моделирование и т. п.) и представить эту информацию в стандартном виде. На практике, конечно, намного правильнее было бы перенести выполнение расчетов (группировку, вычисление итогов по группам и т. п.) на сервер баз данных, но обычно у пользователей для этого нет ни необходимых знаний, ни прав для работы с сервером баз данных. Поэтому Excel в таких ситуациях остается незаменимым средством;
- вариант первой ситуации — приложение, работающее с базой данных, уже умеет генерировать отчеты в формате файлов Excel. Но со временем потребности в отчетах изменяются, появляется необходимость в новых отчетах или в изменении старых. Чаще всего в этом случае пользователи самостоятельно создают новые отчеты, используя данные из старых. Повторяющихся действий очень много, поэтому автоматизация таких операций бывает просто необходима;
- очень часто пользователи, не имея возможности обратиться к профессиональным программистам, самостоятельно реализуют нужные им приложения в таблицах Excel. Во множестве организаций, например, финансовое планирование или составление смет ведется просто в виде множества

файлов Excel (часто связанных между собой). Excel выполняет и роль базы данных, и роль клиентского приложения, и генератора отчетов. В таких ситуациях, конечно, опять-таки вопросы автоматизации стоят очень остро;

- формат файлов Excel удобен не только для вывода информации из базы данных, но и для загрузки введенной вручную информации в базу данных. Часто на предприятиях информация из филиалов, подразделений, от сотрудников и т. п. собирается в формате Excel. В результате со временем возникает вопрос — как автоматизировать процесс загрузки информации из Excel в базу данных;
- по моему опыту, на предприятиях часто возникает потребность в синхронизации информации между файлами Excel и базами данных (или другими файлами Excel, или файлами DBF и т. п.). Например, нужно сделать так, чтобы при занесении пользователем информации в файл Excel она сразу же добавлялась в базу данных.

Приемы, необходимые для решения подобных задач, рассматриваются в данной главе. Надеемся, что после ее изучения у вас не возникнет проблем с тем, как их решать.

С программной точки зрения Excel, в отличие от Word, чаще всего используется не для вывода и редактирования данных, а для выполнения различных расчетов и отображения их в специальных форматах (график, сводная таблица и т. п.). Если же объем данных большой (например, нужно хранить информацию по заказчикам, договорам или поставкам, то имеет смысл подумать о связке Excel плюс база данных (такая связка может быть очень удобной и производительной).

По сравнению с программным перемещением по документам Word навигацию по книгам и листам Excel производить намного удобнее, поскольку у каждой ячейки есть свой адрес (и даже два адреса — в формате A1 и в формате R1C1). Кроме того, в Excel есть возможность присваивать имена диапазонам ячеек, что также очень удобно.

Иерархия стандартных объектов в Excel немного больше. Если в Word все построено вокруг трех объектов: `Application` — `Document` — `Range`, то в Excel появляется новый элемент — лист, поэтому главная его иерархия выглядит следующим образом: `Application` — `Workbook` (книга) — `Worksheet` (лист) — `Range` (диапазон).

В Excel предусмотрена очень богатая библиотека встроенных функций (статистических, финансовых, математических и т. п.), которые можно использовать в приложениях. Часто именно наличие такой библиотеки функций оказывается решающим при выборе Excel в качестве платформы для построения приложения.

В Excel встроено несколько фактически внешних приложений, использование которых может быть очень удобным. Например, сводная таблица (объект `PivotTable`) — интегрированный в Excel OLAP-клиент приобретенной Microsoft фирмы Panorama Software, `QueryTable` — специальный объект для работы с информацией из базы данных, объект `Chart` — средство работы с диаграммами и т. п.

11.2. Объект *Application*

Как и в Word, объект `Application` в Microsoft Excel представляет все приложение Excel и находится на самом верхнем уровне объектной модели Excel. Если вам потребуется вызвать Excel из другого приложения, вам нужно будет создать объект `Excel.Application` (не забудьте при этом при помощи меню **Tools | References** добавить ссылку на библиотеку Microsoft Excel 11.0 Object Library). Создание этого объекта может выглядеть так:

```
Dim oExcel As New Excel.Application
oExcel.Workbooks.Add
oExcel.Visible = True
```

Точно так же, как и в Word, если вы работаете из уже запущенного Excel, создавать объект `Application` вам не потребуется. Он будет доступен всегда. Если вы обращаетесь к какому-либо свойству без указания вышестоящего объекта, то редактор Visual Basic в Excel будет считать, что вы обращаетесь к свойству объекта `Application`. Поэтому эти две строки кода в Excel равнозначны:

```
Application.Workbooks.Add
```

и

```
Workbooks.Add
```

Вообще объекты `Application` в большинстве приложений Office очень похожи между собой, и к ним применяются те же соображения, что и для объекта `Word.Application`. Точно так же многие разработчики считают, что удобнее и надежнее работать со скрытым окном Excel, что открывать новый экземпляр Excel удобнее, чем разыскивать уже открытый пользователем. Для того чтобы в окне редактора кода для форм появился объект `Application`, точно так же необходимо в разделе `Declarations` кода формы объявить объект `Application` с ключевым словом `WithEvents`, например, так:

```
Public WithEvents App As Excel.Application
```

В этом случае в окне редактора кода для форм у вас появится новый объект `App`, и вы сможете использовать событийные процедуры объекта `Application` (рис. 11.1).

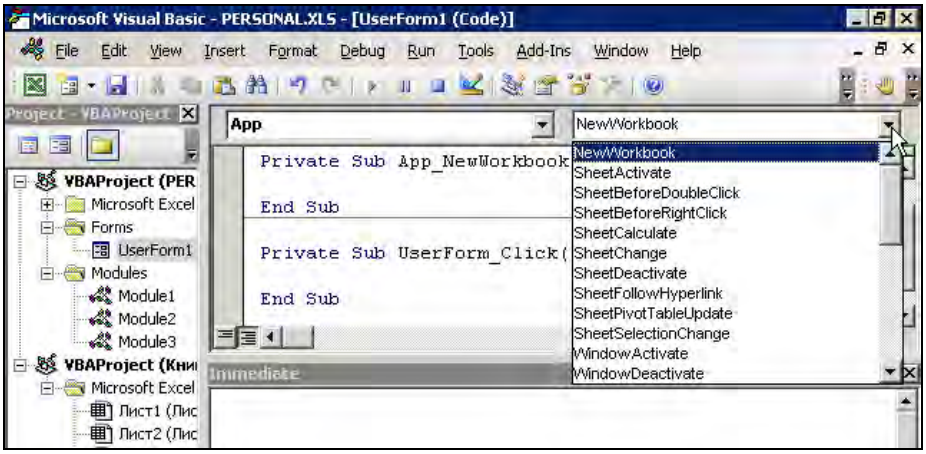


Рис. 11.1. В списке объектов появился новый объект `App` со своим набором событий

11.3. Свойства и методы объекта *Application*

Многие свойства, методы и события объекта `Excel.Application` совпадают с `Word.Application`. Однако т. к. здесь информация приводится для справки для тех пользователей, которым трудно читать по-английски, приведу наиболее часто используемые свойства и методы объекта `Application` в Excel, вне зависимости от того, встречались ли они нам в Word или нет.

Вначале о свойствах объекта `Application`.

- Свойства с префиксом `Active...` — возвращают активную ячейку (ту, на которую указывает курсор ввода данных), активную диаграмму, активный лист, активную книгу или активное окно. Все эти свойства доступны только для чтения. Собственно говоря, использовать их для создания объектов совсем не обязательно — объекты `ActiveCell`, `ActiveSheet` и т. п. создаются автоматически во время работы приложения и доступны всегда. Немного отличается свойство `ActivePrinter` — оно позволяет не только вернуть, но и установить активный принтер.
- `AddIns` — возвращает одноименную коллекцию надстроек (объектов `AddIn`). В отличие от Word, где в большинстве случаев применение надстроек предназначено для профессиональных программистов, в Excel работа с этим объектом имеет практическое значение для многих пользователей. Вместе с Excel поставляется несколько очень полезных надстроек (на графическом экране они доступны через меню **Сервис | Надстройки**), например, **Мастер подстановок**, **Пакет анализа**, **Поиск решения** и т. п. При помощи этой коллекции можно проверить, подключена ли пользова-

телем нужная надстройка (если она нужна в вашей программе) и в случае необходимости подключить ее автоматически.

- `AutoRecover` — возвращает одноименный объект, который позволяет определить параметры автосохранения Excel. Например, чтобы открытые документы Excel автоматически сохранялись каждые 5 минут, можно использовать код:

  ```
  Application.AutoRecover.Time = 5
  ```

  Время указывается в минутах, можно использовать значения в интервале от 1 до 120. На графическом экране то же самое можно сделать при помощи меню **Сервис | Параметры** на вкладке **Сохранение** окна **Параметры**.

- `Calculation` — позволяет узнать или настроить режим пересчета рабочей книги (по умолчанию установлен автоматический режим, можно также использовать ручной пересчет или полуавтоматический, когда автоматически пересчитывается все, кроме таблиц). Есть смысл отключать автоматический пересчет тогда, когда пересчет значений после каждого изменения ячейки занимает много времени и мешает вводу данных. То же самое на графическом экране можно настроить при помощи меню **Сервис | Параметры**, вкладка **Вычисления** окна **Параметры** (явно дать команду на пересчет можно клавишей <F9>).

- `CalculationState` — позволяет проверить, занимается ли Excel пересчетом данных или пересчет уже завершен.

- `Cells` — одно из самых важных свойств объекта `Application`. Оно возвращает объект `Range`, представляющий собой все ячейки в активном листе активной книги. Поскольку свойство по умолчанию (т. е. свойство, название которого можно опускать) для объекта `Range` — это свойство `Item`, то обращение к ячейкам активного листа может выглядеть так:

  ```
  Application.Cells(1, 2).Font.Bold = True
  ```

  В данном случае мы выделили полужирным ячейку на пересечении первой строки и второго столбца.

  Очень похоже действуют свойства `Columns` и `Rows`. Например, чтобы проделать подобную операцию со всем вторым столбцом, можно использовать команду вида:

  ```
  Application.Columns(2).Font.Bold = True
  ```

  а для второй строки можно воспользоваться похожей командой:

  ```
  Application.Rows(2).Font.Bold = True
  ```

  Еще раз отметим, что свойства `Cells`, `Columns` и `Rows` возвращают вовсе не наборы объектов `Cell`, `Column` и `Row`, как считают многие пользователи, а

наборы объектов Range. На использовании объекта Range построена в Excel почти вся работа с ячейками и их значениями. Далее в этой главе объекту Range будет посвящен отдельный *разд. 11.6*.

- Cursor — это свойство позволяет поменять внешний вид указателя мыши в Excel (у объекта Application в Word этого свойства почему-то нет). Обычно перед выполнением длинной расчетной операции курсору придают вид песочных часов (xlWait), а потом возвращают обратно. Автоматически по завершению работы макроса курсор не возвращается к нормальному виду, поэтому нужно предусмотреть соответствующий код.

- DataEntryMode — очень интересное свойство, которое может уберечь пользователя от множества ошибок. Оно позволяет перейти в режим ввода данных *(data entry mode)*, когда пользователю разрешается только вводить данные в разблокированные ячейки выбранного диапазона. Всего в вашем распоряжении три варианта: xlOn — включить этот режим, xlStrict — включить и сделать так, чтобы пользователь не мог из него выйти при помощи клавиши <Esc>, xlOff — отключить режим.

- DecimalSeparator и ThousandsSeparator — эти свойства позволяют не полагаться на региональные настройки на компьютере пользователя, а явно назначить символы, которые будут отделять дробную часть числа от целой и тысячи друг от друга. При использовании этих свойств рекомендуется также отключить использование системных установок при помощи свойства UseSystemSeparators:

```
Application.UseSystemSeparators = False
```

- Dialogs — возвращает одноименную коллекцию Dialog, которую можно использовать для отображения диалоговых окон Excel (их предусмотрено несколько сотен) и определять реакцию на действия в них пользователей. На этот объект очень похож объект FileDialog, представляющий окна, предназначенные только для работы с файлами (например, окно открытия файла). Работа с ними выглядит точно так же, как в Word.

- DisplayAlerts — свойство, про которое мы уже говорили в модуле про Word, но по причине его большой важности повторим еще раз. Это свойство позволяет отключить показ различных предупреждений, которые пользователю обычно в ходе работы приложения показывать не надо (например, подавить предупреждение при закрытии ненужного файла, в котором не были сохранены изменения).

- EnableEvents — позволяет на время отключить события для объекта Application, чтобы они не срабатывали (обычно перед выполнением какого-то действия — открытия файла, сохранения и т. п.).

- `ErrorCheckingOptions` — возвращает ссылку на одноименный объект, при помощи свойств которого можно настроить параметры автопроверки Excel (сообщать ли пользователю о синтаксически неверных формулах, ссылках на пустые ячейки и т. п.). По умолчанию большинство проверок включено, и к этому объекту есть смысл обращаться только тогда, когда вы хотите их отменить.
- `FileDialog` — позволяет обратиться к диалоговым окнам открытия и сохранения файлов (то же самое можно сделать при помощи более общего свойства `Dialogs`). Например, чтобы предоставить пользователю выбрать единственный файл в окне открытия и получить полный путь к нему, можно воспользоваться кодом:

  ```
  Application.FileDialog(msoFileDialogOpen).AllowMultiSelect = False
  Application.FileDialog(msoFileDialogOpen).Show
  Debug.Print Application.FileDialog(msoFileDialogOpen).SelectedItems(1)
  ```

  Похожий пример с возможностью выбора сразу нескольких файлов приведен в справке по этому свойству.
- `FileSearch` — это свойство позволяет провести поиск по указанному вами каталогу и вывести результат.
- `Interactive` — позволяет полностью заблокировать ввод в приложение Excel со стороны пользователя (как клавиатуру, так и мышь). Обычно используется, чтобы пользователь не смог помешать работе приложения, например, сбить выделение. Это свойство можно также использовать, если ввод пользователя производится из другого приложения, взаимодействующего с Excel.
- `International` и `LanguageSettings` — работают точно так же, как и в Word.
- `LibraryPath` — возвращает путь к каталогу, где лежат файлы надстроек Excel с расширением xla. По умолчанию \Office11\Library.
- `MoveAfterReturn` — позволяет включить или отключить переход на следующую ячейку после завершения ввода данных и нажатия <Enter> (по умолчанию включен), а свойство `MoveAfterReturnDirection` позволяет определить направление перехода. В некоторых ситуациях это может сильно упростить ввод данных пользователем. Например, чтобы переход происходил на ячейку справа, можно использовать команду:

  ```
  Application.MoveAfterReturnDirection = xlToRight
  ```

- `Names` — возвращает коллекцию `Names`, представляющую собой все именованные диапазоны в активной рабочей книге. При помощи метода `Add()` коллекции `Names` вы можете также сами определять в рабочей книге свои именованные диапазоны. На практике именованные диапазоны работают

примерно так же, как закладки в Word — с их помощью очень удобно определять наборы данных в сложных таблицах Excel. На графическом экране в Excel определить именованные диапазоны можно при помощи меню **Вставка | Имя**.

- `ODBCErrors` и `OLEDBErrors` — позволяют получить информацию о возникших ошибках при подключении к базам данных ODBC и OLE DB соответственно. Они возвращают одноименные коллекции, которые состоят из объектов `ODBCError` и `OLEDBError` соответственно, которые и содержат информацию об ошибке.
- `OnWindow` — это свойство больше похоже на событие. В качестве его значения указывается имя процедуры, которая должна находиться в модуле уровня книги (по умолчанию такой модуль не создается, его нужно создать вручную). Эта процедура будет вызываться всякий раз, когда пользователь переключился в окно Excel (не важно какой книги и какого листа). Вместо этого свойства можно использовать макросы со специальными именами `Auto_Activate` и `Auto_Deactivate` (если вы определили и то, и другое, первой отработает процедура, определенная при помощи свойства `OnWindow`).
- `Range` — очень важное свойство. Возвращает объект `Range`, который представляет собой диапазон ячеек и используется в Excel практически для любых операций с ячейками.
- `ReferenceStyle` — позволяет переключать режим отображения ячеек между A1 (буквы — столбцы, цифры — строки) и R1C1 (когда и строки, и столбцы обозначаются цифрами). На графическом экране это можно сделать через меню **Сервис | Параметры** на вкладке **Общие** окна **Параметры**, установив или сбросив флажок **Стиль ссылок R1C1**. На практике пользователям ближе стиль вида A1, а программистам, конечно, R1C1 (особенно в тех ситуациях, когда столбцов очень много и приходится использовать столбцы AA, AB и т. п.). Во многих ситуациях перед выполнением какой-то программной операции бывает удобно вначале перевести режим отображения в R1C1, а после окончания на радость пользователям вернуть его обратно. Можно этим и не заниматься, а использовать другие способы для отсчета определенного количества столбцов.
- `Selection` — как несложно догадаться, это свойство возвращает то, что в настоящий момент выбрал пользователь. Если он выбрал обычные ячейки в таблице, то вернется объект `Range`. Если же пользователь выбрал что-то на диаграмме, то может вернуться объект осей, легенды и т. п., в зависимости от того, что было выбрано.
- `Sheets` — это свойство мы будем подробнее разбирать в *разд. 11.5*, посвященном книгам Excel. Оно возвращает коллекцию `Sheets` — набор листов

книги и набор диаграмм, которые находятся на отдельных листах. Если используется свойство `Worksheets`, то вернется та же коллекция `Sheets`, но уже состоящая только из объектов `Worksheet` — обычных листов (без диаграмм).

- `TemplatesPath` — свойство для чтения, при помощи которого можно получить информацию о каталоге с шаблонами Excel (например, для размещения собственного шаблона или открытия шаблона из этого каталога). По умолчанию используется каталог Application Data\Microsoft\Templates в профиле пользователя.
- `ThisCell` и `ThisWorkbook` — очень удобные свойства, которые позволяют обращаться к текущей ячейке и к текущей книге, не обременяя себя созданием объектных переменных. Эти объекты создавать не нужно — они и так изначально существуют в работающем Excel.
- `Windows`, `Workbooks` и `Sheets` — возвращают, соответственно, все открытые окна, книги и листы Excel. Про объекты рабочей книги и листа мы будем говорить в *разд. 11.5*.
- `WorkSheetFunction` — позволяет использовать в программе на VBA функции Excel напрямую, без необходимости прописывать их в какую-либо ячейку.

Самые важные методы объекта `Excel.Application` перечислены далее.

- `ActivateMicrosoftApp()` — специальный метод, который предназначен для запуска и активизации (или просто активизации, если приложение уже было запущено) приложений Office (Word, Access, PowerPoint) и некоторых других (Project, FoxPro, Schedule Plus).
- `AddCustomList()` — создает новый пользовательский список. В качестве параметра принимает либо объект `Range` (элементами списка станут те значения, которые есть в диапазоне), либо стандартный объект `Array`. Удалить список можно при помощи метода `DeleteCustomList()`.
- Методы с префиксом `Calculate...` — позволяют пересчитать значения в ячейках. Простой метод `Calculate()` — это обычный пересчет значений (чаще всего нужен, если автопересчет отключен), `CalculateFull()` пересчитывает значения во всех открытых книгах, `CalculateFullRebuild()` — еще и производит перестроение формул (аналогично занесению всех формул заново). Остановить пересчет можно при помощи метода `CheckAbort()`.
- `ConvertFormula()` — преобразовывает формулу двумя способами: либо переводит адресацию ячеек в другой режим (например, вместо A1 в R1C1), либо меняет абсолютные ссылки на относительные и наоборот. В качестве параметра принимает строковую переменную с текстом формулы (должна начинаться с символа '=') и флаги конвертации.

- `DoubleClick()` — этот метод эквивалентен двойному щелчку мышью на активной ячейке, т. е. переход в режим ввода данных в эту ячейку.
- `Evaluate()` — очень полезный и часто используемый метод. Позволяет по имени найти объект книги Excel и преобразовать его в объект или значения для дальнейшего использования. В качестве имен этот метод может принимать:
  - имена ячеек в стиле A1 (возвращается объект `Cell`);
  - имена диапазонов (возвращается объект `Range`);
  - имена, определенные в макросе (чаще всего названия переменных);
  - ссылки на внешние книги, например:

    ```
    Evaluate("[Book1.xls]Sheet1!A5"))
    ```

  Этот метод можно вызвать и неявно, просто заключив имя объекта в квадратные скобки. Например, такие строки будут абсолютно одинаковыми по значению:

  ```
  [a1].Value = 25
  Evaluate("A1").Value = 25
  ```

  Поскольку синтаксис с квадратными скобками короче, чаще всего используется именно он.

  Таким образом, метод `Evaluate()` — это самый простой и естественный метод обратиться к ячейке или диапазону в своей или чужой книге Excel.
- `GetOpenFilename()` — это упрощенный способ работы с окном открытия файлов. Чтобы открыть диалоговое окно и получить информацию о том, что выбрал пользователь (в виде строковой переменной с информацией о имени файла с полным путем), можно использовать такой код:

  ```
  Filename = Application.GetOpenFilename()
  If Filename <> False Then
      Debug.Print Filename
  End If
  ```

- `GetSaveAsFilename()` — такой же по функциональности способ, который работает с окном **Сохранить как**.
- `GoTo()` — важный и часто используемый метод. Принимает в качестве параметра объект `Range`, строку со ссылкой на ячейку (в формате R1C1) или имя процедуры VBA, выделяет и активизирует диапазон или ячейку или запускает на выполнение процедуру. В отличие от `Select()`, этот метод еще и автоматически активизирует лист, на котором расположены диапазон или ячейка.

- `Help()` — позволяет открыть и показать информацию из файла справки (в том числе пользовательского). В качестве параметра принимает имя файла справки и метку темы в нем.
- `Intersect()` — возвращает диапазон, который представляет собой область пересечения двух или более диапазонов. Если передаваемые в качестве параметров диапазоны не пересекаются, возвращается `Nothing`.
- `OnKey()` — этот метод позволяет "посадить" процедуру VBA на определенную клавиатурную комбинацию. Например, чтобы назначить процедуру `Msg()` из модуля `Лист1` клавиатурной комбинации <Alt>+<M>, можно воспользоваться командой:

```
Application.OnKey "%{m}", "Лист1.Msg"
```

- `OnRepeat()` — позволяет назначить процедуру, которая будет выполняться при использовании команды **Повторить** в меню **Правка**. Назначение процедуры команде **Отменить** в том же меню производится при помощи метода `OnUndo()`.
- `RegisterXLL()` — подключает файл надстройки Excel с расширением xll и регистрирует его функции и процедуры. Этот метод используется при установке собственного приложения.
- `Repeat()` — позволяет просто повторить последнее действие, которое было выполнено пользователем (не программным способом). Аналогично команде **Повторить** в меню **Правка**. Отменить действие пользователя можно при помощи метода `Undo()`.
- `Run()` — позволяет выполнить процедуру или функцию VBA, макрос Excel или процедуру или функцию в XLL-модуле (и передать до 30 параметров).
- `SendKeys()` — позволяет эмулировать нажатия клавиш и передать их в активное окно приложения. Обычно используется, если нужно что-то продемонстрировать пользователю или что-то сделать через меню (например, не удалось найти, как это можно выполнить с помощью кода).
- `Union()` — позволяет объединить два или более диапазонов.
- `Volatile()` — этот хитрый метод должен вызываться только из пользовательской функции, которая вычисляет значение ячейки. Если он вызван и ему передано значение `True`, то данная ячейка становится чувствительной *(volatile)* и будет пересчитываться при любом изменении значений в листе, даже том, которое ее не касается.
- `Wait()` — этот метод позволяет приостановить работу Excel на указанное вами время, сняв нагрузку с процессора. Используется для демонстраций, чтобы пользователь успел увидеть, что происходит, для ожидания завершения выполнения какой-либо внешней операции и т. п. При этом ввод

пользователя блокируется, а указатель мыши приобретает вид песочных часов. Однако некоторые фоновые операции Excel, такие как печать и пересчет значений, будут продолжать выполняться. Например, чтобы взять паузу на пять секунд, можно воспользоваться кодом:

```
If Application.Wait(Now + TimeValue("0:00:5")) Then
    MsgBox "Пять секунд прошло"
End If
```

- `Quit()` и `OnTime()` — делают то же самое, что и в Word.

11.4. Коллекция *Workbooks* и объект *Workbook*, их свойства и методы

Следующий по иерархии после `Application` объект в объектной модели Excel — это объект `Workbook`, который представляет собой книгу Excel. Можно сказать, что объект `Workbook` занимает в Excel примерно то же место, что и объект `Document` в Word — он необходим для получения ссылки на нужную нам книгу в наборе открытых книг Excel, для настройки общих свойств и выполнения общих действий со всеми листами книги. Получить этот объект можно очень просто:

- первый способ — воспользоваться коллекцией `Workbooks`, которая доступна через свойство `Workbooks` объекта `Application`. Впрочем, применять это свойство совершенно не обязательно — коллекция `Workbooks` в Excel и так постоянно доступна. Найти нужную книгу в этой коллекции можно по ее имени или номеру в коллекции, например:

  ```
  Debug.Print Workbooks("Смета.xls").FullName
  ```

- второй способ — использовать свойство `Application.ActiveWorkbook`. При помощи этого свойства мы обращаемся к активной в настоящей момент книге:

  ```
  Debug.Print ActiveWorkbook.Name
  ```

- третий способ — использовать свойство `Application.ThisWorkbook`. При этом мы обращаемся к книге, которой принадлежит данный программный модуль:

  ```
  Debug.Print ThisWorkbook.Name
  ```

На практике чаще всего нам нужно либо создать в Excel новую книгу, либо открыть существующую книгу (или другой файл в формате, который понимает Excel, например, DBF). Для этой цели используются методы `Add()` и `Open()` соответственно. Например, создать новую книгу в Excel можно так:

```
Dim oWbk As Workbook
Set oWbk = Workbooks.Add()
```

Единственный необязательный параметр, который принимает этот метод, — имя шаблона, на основе которого создается новая рабочая книга.

Открытие существующей книги выглядит так:

```
Dim oWbk As Workbook
Set oWbk = WorkBooks.Open("C:\mybook1.xls")
```

Кроме стандартных, в коллекции `Workbooks` предусмотрено также три специальных метода.

- `OpenDatabase()` — открывает базу данных, выполняет запрос к ней (или напрямую открывает таблицу или представление), а результаты запроса помещает как импортированные внешние данные в новую автоматически созданную рабочую книгу Excel;
- `OpenText()` — почти то же самое, но в качестве источника здесь выступает текстовый файл. Дополнительные параметры позволяют определить его формат.
- `OpenXML()` — в качестве источника данных будет выступать файл в формате XML.

Как и метод `InsertDatabase()` в Word, эти методы следует использовать только в самых простых случаях. Рекомендуется по возможности применять более мощные и стандартные средства объектной модели ADO.

Теперь о самых важных свойствах объекта `Workbook` — самой рабочей книги.

- `Name`, `CodeName`, `FullName` — разные имена этой книги. Самое простое имя — `Name`, которое совпадает с именем файла книги. `FullName` — это имя файла книги вместе с полным путем к нему в операционной системе. `CodeName` — как эта книга называется в коде. `CodeName` можно посмотреть в окне **Project Explorer** или, если открыть свойства книги в окне **Properties**, кодовое имя книги будет представлено в строке **(Name)**. Все три свойства доступны только для чтения, менять их можно другими способами (например, сохраняя файл под другим именем или изменив свойство в окне **Properties**).

  Определенное отношение к именам имеет также свойство `Path`, которое возвращает полный путь в файловой системе к книге Excel.

- `Charts`, `Sheets`, `ActiveChart`, `ActiveSheet`, `CustomViews`, `BuiltinDocumentProperties` и `CustomDocumentProperties`, `Windows`, `WebOptions` — возвращают одноименные коллекции соответствующих объектов. Некоторые из них будут рассматриваться в следующих разделах.

- `ConflictResolution` — определяет, как будут разрешаться конфликты изменения данных, если книга открыта сразу несколькими пользователями *(shared workbook)*. Есть возможность сделать так, чтобы локальный пользователь автоматически выигрывал, автоматически проигрывал или возникало диалоговое окно с возможностью разобраться в конфликте вручную. Существует большое количество свойств, которые позволяют настроить параметры совместной работы с книгой, но по причине того, что такая работа не рекомендуется (данные для совместного доступа необходимо переносить в базу данных), рассматриваться они здесь не будут, за исключением:
  - запрещать или разрешать общий доступ к рабочей книге можно при помощи методов `SaveAs()` или `ExclusiveAccess()`;
  - по умолчанию возможность совместного редактирования для книги отключена (проверить можно при помощи свойства `MultiUserEditing`);
  - получить список всех пользователей (а также информацию, когда они открыли файл и в каком режиме) можно при помощи свойства `UserStatus`.
- `FileFormat` — возвращает формат книги (доступен напрямую только для чтения, можно изменять при сохранении книги). Форматов очень много: разные версии Excel, DBF, Lotus 1-2-3, форматы TXT, CSV, XML — всего несколько десятков.
- `Names` — возвращает коллекцию всех именованных диапазонов в данной рабочей книге. Получить информацию о таких диапазонах можно, например, так:

```
For Each Item In ThisWorkbook.Names
    Debug.Print Item.Name
Next
```

  Это свойство удобно использовать для предварительных проверок для устранения потенциальных ошибок времени выполнения.

Методов у объекта `Workbook` также очень много, однако самые часто используемые из них — это `Activate()`, `Close()`, `Save()`, `SaveAs()`, `PrintOut()`, `Protect()` и `Unprotect()`, которые очевидны и действуют аналогично одноименным методам объекта `Document` в Word.

11.5. Коллекция *Sheets* и объект *Worksheet*, их свойства и методы

В Word на уровне ниже объекта `Application` и `Document` начинались уже объекты непосредственно для работы с текстом: `Selection`, `Range` и т. п. В Excel

между объектом рабочей книги и ячейками есть еще один промежуточный объект — `Worksheet` (лист). Объекты `Worksheet` в книге объединены в коллекцию `Sheets`.

Чаще всего для ввода данных в Excel (напрямую или из базы данных) нам потребуется, в первую очередь, определиться с листом, на который будет выполняться ввод данных — либо просто выбрать его, либо вначале создать, а потом выбрать.

Процесс создания нового листа выглядит очень просто:

```
Dim oExcel As New Excel.Application     'Запускаем Excel
oExcel.Visible = True                 'Делаем его видимым
Dim oWbk As Excel.Workbook
Set oWbk = oExcel.Workbooks.Add()    'Создаем новую книгу
Dim oSheet As Excel.Worksheet
Set oSheet = oWbk.Worksheets.Add()  'Создаем новый лист
oSheet.Name = "Новый лист"           'Присваиваем ему имя "Новый лист"
```

Метод `Add()` для коллекции `Worksheets` может принимать несколько необязательных параметров, главная задача которых — определить, между какими существующими листами книги будет вставлен новый лист. Если ничего не указывать, то новый лист будет помещен самым первым.

Часто встречается и другая задача — найти нужный лист среди листов книги, например, если мы открыли существующую книгу. Сделать это очень просто, поскольку коллекция `Worksheets` умеет работать с именами листов. Далее приведен пример, в котором мы запускаем Excel и создаем новую книгу, но при этом находим лист с именем `"Лист1"` и переименовываем его в `"Новый лист"`:

```
Dim oExcel As New Excel.Application         'Запускаем Excel
oExcel.Visible = True                       'Делаем его видимым
Dim oWbk As Excel.Workbook
Set oWbk = oExcel.Workbooks.Add()           'Создаем новую книгу
Dim oSheet As Excel.Worksheet
Set oSheet = oWbk.Worksheets.Item("Лист1")  'Находим Лист1
oSheet.Name = "Новый лист"      'Присваиваем ему имя "Новый лист"
```

Обратите внимание, что в английской версии Excel этот код не пройдет, поскольку листы там по умолчанию называются `"Sheet1"`, `"Sheet2"` и т. п. Если вы используете в коде имена листов, заданные по умолчанию, и при этом вашей программе придется работать на компьютерах с разноязычными версиями Excel, обязательно предусмотрите дополнительные проверки или просто используйте номера листов вместо их имен.

У коллекции `Sheets` есть привычные нам свойства и методы коллекций VBA (`Count`, `Item`, `Add()`, `Delete()`). Другие свойства и методы удобнее применять для объекта `Worksheet` (`Visible()`, `Copy()`, `Move()`, `PrintOut()`, `PrintPreview()`, `Select()`), поскольку все равно нам придется указывать конкретный лист. Однако для этой коллекции предусмотрен и один специфический метод `FillAcrossSheets()` — скопировать объект диапазона `Range` (полностью, только содержимое или только оформление) во все листы данной книги.

У объекта `Worksheet` есть множество важных свойств и методов.

- `Cells` — одно из наиболее часто используемых свойств. Работает точно так же, как и рассмотренное ранее одноименное свойство объекта `Application`, за исключением того, что вам не придется ограничиваться только активным листом. Аналогично работают свойства `Columns` и `Rows`.
- `EnableCalculation` — позволяет отключить автоматический пересчет значений ячеек на листе.
- `EnableSelection` — позволяет запретить выделять что-либо на листе, снять запрет или разрешить выделять только незаблокированные ячейки.
- `Next` — получает ссылку на следующий лист в книге, в свойство `Previous` — на предыдущий лист.
- `PageSetup` — как и в Word, позволяет получить объект `PageSetup`, при помощи которого можно настроить те же параметры, что и через меню **Файл | Параметры страницы**.
- `Protection` — позволяет получить объект `Protection`, при помощи которого можно запретить пользователю вносить изменения в лист Excel. Для настройки параметров защиты предназначены также и другие свойства, названия которых начинаются с префикса `Protection...`
- `QueryTables` — исключительно важное свойство. Оно возвращает коллекцию `QueryTables` — набор объектов `QueryTable`, которые, в свою очередь, представляют данные, полученные из внешних источников (как правило, из баз данных).
- `Range` — самое важное свойство объекта `Worksheet`. Возвращает объект `Range` (диапазон ячеек), который в объектной модели Excel занимает примерно такое же место, что и одноименный объект в объектной модели Word. Этот объект будет рассматриваться далее в *разд. 11.6*.
- `Type` — определяет тип данного листа. Обычно используются два типа: `xlWorksheet` (обычный лист) и `xlChart` (диаграмма).
- `UsedRange` — возвращает объект `Range`, представляющий собой прямоугольную область, включающую все непустые ячейки листа. Удобно использовать для копирования или форматирования.

- `Visible` — позволяет спрятать лист от пользователя (например, если он используется для служебных целей).

Некоторые важные методы объекта `Worksheet` представлены далее.

- `Activate()`, `Calculate()`, `Copy()`, `Paste()`, `Delete()`, `Move()`, `Evaluate()`, `Select()`, `SaveAs()`, `PrintOut()`, `PrintPreview()`, `Protect()`, `Unprotect()` — эти методы нам уже знакомы. Отличие заключается только в том, что теперь эти методы могут применяться для выбранного вами листа.
- `PivotTables()` — возвращает коллекцию очень интересных объектов `PivotTable` (сводная таблица), которые будут рассматриваться в *разд. 11.8*.
- `Scenarios()` — возвращает коллекцию `Scenarios`, состоящую из объектов `Scenario` (сценарии). Сценарии — это именованные наборы вводных данных, которые можно использовать для проверки различных вариантов (разные суммы продаж, уровни налогов, расходов и т. п.).
- `SetBackgroundPicture()` — позволяет назначить листу фоновое изображение (естественно, желательно, чтобы оно было полупрозрачным, как "водяной знак", иначе на его фоне будет трудно читать текст в ячейках).
- `ShowAllData()` — показывает все скрытые и отфильтрованные данные на листе.

Самое важное событие объекта `Worksheet` — это, конечно, `Change`. Существует множество практических задач, когда изменение пользователем значения в ячейке должно приводить к изменению значения в ячейке другого листа или рабочей книги Excel, или даже в базе данных. Другая ситуация, в которой используется это событие, — сложная проверка вводимого пользователем значения (например, когда это значение сверяется со значением в базе данных). Эта событийная процедура работает со специальным параметром `Target`, т. е. с объектом `Range`, представляющим изменившуюся ячейку. При помощи свойств и методов объекта `Range` вы можете получить информацию об изменившемся значении, столбце и строке, в котором произошло изменение, и т. п.

У объекта `Worksheet` есть еще два очень удобных события (их сильно не хватает объекту `Document` в Word) — это `BeforeRightClick()` и `BeforeDoubleClick()`. Как понятно из названий, первое событие позволяет перехватывать щелчок правой кнопкой мыши по любому месту в листе, а второе событие — двойной щелчок мышью. При помощи этих событий вы можете назначить свою реакцию (открытие контекстных меню, выдачу предупреждающих сообщений, переход в другой режим работы и т. п.) на действия пользователя.

11.6. Объект *Range*, его свойства и методы

Пожалуй, наиболее часто используемый объект в иерархии объектной модели Excel — это объект `Range`. Он может представлять одну ячейку, несколько ячеек (в том числе несмежные ячейки или наборы несмежных ячеек) или целый лист. Если в Word вы могли для ввода данных использовать как объект `Range`, так и объект `Selection`, то в Excel все сводится к объекту `Range`:

- если вам нужно ввести данные в ячейку или отформатировать ее, то вы должны получить объект `Range`, представляющий эту ячейку;
- если вы хотите сделать что-то с выделенными вами ячейками, вам необходимо получить объект `Range`, представляющий выделение;
- если вам нужно просто что-то сделать с группой ячеек, первое ваше действие — это получить объект `Range`, представляющий эту группу ячеек.

В базе знаний Microsoft (**www.microsoft.com/support**) есть статья под номером 291308, в которой описываются 22 способа получения объекта `Range` в Excel. Вряд ли вы будете пользоваться всеми эти способами. Мы рассмотрим только самые распространенные.

- Самый простой и очевидный способ — воспользоваться свойством `Range`. Это свойство предусмотрено для объектов `Application`, `Worksheet` и для самого объекта `Range` (если вы решили создать новый диапазон на основе уже существующего). Например, получить ссылку на объект `Range`, представляющий ячейку A1, можно так:

```
Dim oRange As Range
Set oRange = Worksheets("Лист1").Range("A1")
```

  А ссылка на диапазон ячеек с A1 по D10 создается так:

```
Dim oRange As Range
Set oRange = Worksheets("Лист1").Range("A1:D10")
```

  С применением свойства `Range` самого объекта `Range` нужно быть очень осторожным. Дело в том, что Excel создает на основе объекта `Range` виртуальный лист со своей собственной нумерацией. Поэтому такой код:

```
Set oRange1 = Worksheets("Лист1").Range("C1")
Set oRange2 = oRange1.Range("B1")
oRange2.Value = 20
```

  пропишет значение 20 не в ячейку B1, как можно было понять из кода, а в ячейку D1 (т. е. в ячейку B1 по отношению к виртуальному листу, начинающемуся с C1).

- Второй способ — воспользоваться свойством `Cells` объекта `Worksheet`. Возможностей у этого свойства меньше — возвращается диапазон, состоящий только из одной ячейки. Зато мы можем использовать более удобный синтаксис (с точки зрения передачи переменных, перехода в любую сторону на любое количество ячеек и т. п.). Например, для получения ссылки на ячейку D1 можно использовать код вида:

  ```
  Dim oRange As Range
  Set oRange = Worksheets("Лист1").Cells(1, 4)
  ```

  Чтобы получить диапазон, состоящий из нескольких ячеек, удобно применять свойства `Range` и `Cells` вместе:

  ```
  Dim oRange
  Set oRange = Range(Cells(1, 1), Cells(5, 3))
  ```

- Третий способ — воспользоваться многочисленными свойствами объекта `Range`, которые позволяют изменить текущий диапазон или создать на основе его новый. Эти свойства будут рассмотрены далее в этом разделе.

Обычно после того, как нужная ячейка найдена, в нее нужно что-то записать. Для этой цели используется свойство `Value`, например:

```
oRange.Value = "Мое значение"
```

Поскольку объект `Range` с функциональной точки зрения очень важен, то свойств и методов у него очень много (и для комфортной работы в Excel их нужно знать). Далее представлены некоторые самые часто употребимые свойства.

- `Address` — позволяет вернуть адрес текущего диапазона. Например, для предыдущего примера вернется $A$1:$C$5. Если в диапазоне только одна ячейка, то вернется значение вида $A$1. Этому свойству можно передать много разных параметров для определения стиля ссылки, абсолютного или относительного адреса для столбцов и строк, по отношению к чему этот адрес будет относительным и т. п. Свойство доступно только для чтения. `AddressLocal` — то же самое, но с поправкой на особенности локализованных версий Excel.

  На практике встречается множество ситуаций, когда адрес ячейки нужно разобрать на части и вернуть из него имя столбца или номер строки. Это очень просто сделать при помощи строковых функций и знака доллара.

  Например, имя столбца для объекта `oRange`, представляющего одну ячейку, можно вернуть так:

  ```
  sColumnName = Mid(oRange.Address,2,(InStr(2,oRange.Address,"$") – 2))
  ```

А номер строки — так:

```
sRowNumber = Mid(oRange.Address,(InStr(2,oRange.Address,"$") + 1))
```

На первый взгляд это кажется сложным, но на самом деле все очень просто — для имени столбца мы просто берем все, что у нас находится между первым знаком доллара (он всегда идет первым символом) и вторым, а для номера строки — все, что находится после второго знака доллара. Найти второй знак доллара можно при помощи встроенной функции `InStr()`, а взять нужное количество символов, начиная с определенной позиции, проще всего при помощи встроенной функции `Mid()`.

- `AllowEdit` — это свойство, доступное только для чтения, позволяет определить, сможет ли пользователь вносить исправления в данную ячейку (набор ячеек) на защищенном листе. Используется для проверок.
- `Areas` — очень важное свойство. Дело в том, что объект `Range` может состоять из несмежных наборов ячеек. Многие методы применительно к таким диапазонам ведут себя совершенно непредсказуемо или возвращают ошибки. Свойство `Areas` позволяет разбить подобные нестандартные диапазоны на набор стандартных. Созданные таким образом объекты `Range` будут помещены в коллекцию `Areas`. Это свойство можно использовать и для проверки "нестандартности" диапазона:

```
If Selection.Areas.Count > 1 Then
    Debug.Print "Диапазон с несмежными областями"
End If
```

- `Borders` — возвращает ссылку на коллекцию `Borders`, при помощи которой можно управлять рамками для диапазона.
- `Cells` — это свойство, как мы уже видели, есть и у объекта `Worksheet`. Для объекта `Range` оно работает точно так же, за исключением того, что опять-таки используется своя собственная виртуальная адресация на основе диапазона:

```
Dim oRange, oRange2 As Range
Set oRange = Range(Cells(2, 2), Cells(5, 3))
Set oRange2 = oRange.Cells(1, 1)  'Вместо A1 получаем ссылку на B2
Debug.Print oRange2.Address       'Проверка
```

Точно такие же особенности есть у свойств `Row` и `Rows`, `Column` и `Columns`.

- `Characters` — это простое с виду свойство позволяет решать непростую задачу: как изменить фрагменты текста или их формат в ячейке, не затрагивая остальные данные. Например, чтобы ввести текст в ячейку A1 и изменить цвет первой буквы, можно воспользоваться кодом:

```
Dim oRange As Range
Set oRange = Range("A1")
oRange.Value = "Мой текст"
oRange.Characters(1, 1).Font.Color = vbRed
```

Если же вам просто нужно изменить значение, то лучше воспользоваться свойством `Value` — как в третьей строке примера.

- `Count` — возвращает количество ячеек в диапазоне. Может использоваться для проверок.
- `CurrentRegion` — очень удобное свойство, которое может пригодиться, например, при копировании или экспорте данных, полученных из внешнего источника (когда нам изначально неизвестно, сколько будет данных). Оно возвращает объект `Range`, представляющий диапазон, окруженный пустыми ячейками (т. е. непустую область, в которую входит исходный диапазон или ячейка). Например, чтобы выделить всю непустую область вокруг активной ячейки, можно воспользоваться кодом:

```
Worksheets("Лист1").Activate
ActiveCell.CurrentRegion.Select
```

Затем, к примеру, можно программным образом создать новый столбец справа от последнего имеющегося, или под последней заполненной строкой создать новую строку с итогами.

- `Dependents` — позволяет получить объект `Range` (скорее всего, включающий несмежные области), представляющий ячейки, которые зависят от ячеек исходного диапазона. Работает только для текущего листа — ссылки во внешних листах этим свойством не отслеживаются. Например, чтобы выделить все ячейки, зависимые от активной, можно использовать код:

```
Worksheets("Лист1").Activate
ActiveCell.Dependents.Select
```

Чтобы просмотреть обратную зависимость, можно использовать свойство `Precedents`. Чтобы просмотреть только первый уровень зависимостей, можно использовать свойства `DirectDependents` и `DirectPrecedents`.

- `End` — еще одно часто используемое свойство. Оно позволяет получить объект `Range`, представляющий последнюю ячейку исходного диапазона. В каком направлении будет возвращаться последняя ячейка, можно определить при помощи передаваемого параметра.
- `Errors` — свойство, которое через коллекцию `Errors` позволяет получить доступ к объектам `Error`, представляющим обнаруженные ошибки в диапазоне.

- `Font` — как и в Word, это свойство позволяет получить доступ к объекту `Font`, при помощи которого можно настроить особенности оформления текста в ячейке (цвет, шрифт, размер букв и т. п.).
- `FormatConditions` — позволяет создать собственный объект, представляющий вариант оформления ячеек, который затем можно применять к разным ячейкам и диапазонам.
- `Formula` — одно из самых важных свойств объекта `Range`. Доступно и для чтения, и для записи. Возвращает текст формулы, прописанной в ячейку (а не вычисленное значение) или позволяет записать формулу в ячейку. Если применить это свойство для диапазона из нескольких ячеек, то формула будет прописана по всей ячейке диапазона. Пример использования этого свойства может выглядеть так:

```
Worksheets("Лист1").Range("A3").Formula = "=$A$1+$A$2"
```

- `FormulaHidden` — позволяет спрятать формулы от пользователя в данном диапазоне. Работает только на защищенных листах.
- `HasFormula` — проверяет диапазон на наличие вычисляемых значений (формул).
- `Hidden` — позволяет спрятать диапазон. Будет работать только в случае, если диапазон включает в себя хотя бы одну строку или столбец целиком, в противном случае вернется ошибка.
- `Interior` — еще одно свойство, связанное с форматированием. Позволяет выделить цветом ячейки диапазона.
- `Item` — возвращает еще один объект `Range`, который определяется путем смещения исходного диапазона.
- `Locked` — позволяет заблокировать ячейки диапазона при защите листа.
- `Name` — позволяет получить ссылку на специальный объект именованного диапазона `Name`. На графическом экране с его возможностями можно познакомиться при помощи меню **Вставка | Имя**. Он позволяет обращаться к диапазонам и формулам по именам и несколько напоминает по функциональности объект `Bookmark` в Word.
- `Next` — позволяет перейти на следующую ячейку. Если лист не защищен, то следующей ячейкой будет считаться ячейка справа, если лист защищен — то следующая незаблокированная ячейка.
- `NumberFormat` — устанавливает один из предопределенных форматов для чисел. Соответствует возможностям вкладки **Число** в меню **Формат | Ячейки** на графическом экране.
- `Offset` — это свойство позволяет получить новый объект `Range` с определенным смещением относительно исходного диапазона. Например, чтобы

получить ячейку со смещением на три ячейки вверх и три ячейки влево, можно использовать код:

```
Worksheets("Лист1").Activate
ActiveCell.Offset(rowOffset:=-3, columnOffset:=-3).Activate
```

- `Orientation` — позволяет сориентировать текст в ячейках. Определяет угол наклона в градусах. Например, чтобы расположить текст по диагонали, можно использовать код:

```
oRange.Orientation = -45
```

- `PageBreak` — это свойство обычно используется для программной вставки разрывов страницы. Его применение может выглядеть так:

```
Worksheets("Лист1").Rows(50).PageBreak = xlPageBreakManual
```

- свойства с префиксом `Pivot...` — относятся к работе с объектом `PivotTable` (сводная таблица). Особенности работы с ним будут рассмотрены в *разд. 11.8*.

- `QueryTable` — это важное свойство позволяет получить ссылку на объект `QueryTable` — полученные с внешнего источника данные, которые находятся в данном диапазоне. Подробнее про объект `QueryTable` будет рассказано в *разд. 11.7*.

- `Range` — это свойство, как уже говорилось ранее, позволяет создать новый диапазон на основе уже существующего. Необходимо помнить про особенности нумерации ячеек в этом случае.

- `Resize` — позволяет изменить текущий диапазон. Например, увеличение его на один столбец вниз и на одну строку вправо можно выполнить так:

```
oRange.Resize(oRange.Rows.Count + 1, oRange.Columns.Count + 1).Select
```

- `ShrinkToFit` — это свойство позволяет автоматически настроить размер текста в диапазоне таким образом, чтобы текст помещался в ширину столбца.

- `Style` — позволяет вернуть объект `Style`, представляющий стиль для указанного диапазона. На графическом экране то, что выполняет объект `Style`, можно сделать через меню **Формат | Стиль**.

- `Text` — получает значение первой ячейки диапазона в виде значения типа `String`. Для объекта `Range` это свойство доступно только для чтения.

- `Validation` — это свойство позволяет вернуть объект `Validation`, при помощи которого можно настроить проверку вводимых в диапазон данных.

- `Value` — наиболее часто используемое свойство объекта `Range`. Позволяет получить или назначить значение (числовое, текстовое или какое-либо

другое) ячейкам диапазона. Точно также используется свойство `Value2`, но с единственным отличием — оно не поддерживает типы данных `Currency` и `Date`.

- `WrapText` — позволяет включить или отключить перевод текста на следующую строку в ячейках диапазона.

Информация о методах объекта `Range` представлена далее.

- `Activate()` — выделяет текущий диапазон и устанавливает курсор ввода на его первую ячейку.
- `AddComment()` — добавляет комментарий к ячейке. Ячейка будет помечена красным уголком, а текст комментария будет показываться в виде всплывающей подсказки. Этот метод можно вызвать только для диапазона, состоящего из одной ячейки. То же самое на графическом экране можно сделать при помощи меню **Вставка | Примечание**.
- `AutoFill()` — позволяет использовать автозаполнение для диапазона (например, если первые две ячейки будут заполнены как 1 и 2, то дальше в автоматическом режиме будет продолжено: 3, 4, 5 и т. п.).
- `AutoFit()` — автоматически поменяет ширину всех столбцов и высоту всех строк в диапазоне, чтобы туда уместился текст ячеек. Можно применять только к тем диапазонам, которые включают в себя хотя бы одну строку или столбец целиком, иначе вернется ошибка.
- `AutoFormat()` — позволяет использовать один из стилей автоформатирования, которые на графическом экране доступны через меню **Формат | Автоформат**.
- `BorderAround()` — позволяет поместить диапазон в рамку с выбранными вами параметрами.
- Методы с префиксом `Clear...` — позволяют очистить содержимое диапазона от значений, форматирования, комментарий и т. п.
- `Consolidate()` — сливает данные нескольких диапазонов (в том числе на разных листах) в один диапазон, используя при этом выбранную вами агрегатную функцию.
- `Copy()` — копирует диапазон. Если место назначения не указано, то он копируется в буфер обмена. Аналогично работает метод `Cut()`, при котором данные исходного диапазона удаляются.
- `CopyFromRecordset()` — очень удобный метод, который позволяет вставить данные из объекта `ADO.Recordset` на лист Excel, начиная с верхнего левого угла указанного диапазона.
- `DataSeries()` — метод, который поможет сэкономить множество времени и избежать возни с функциями даты и времени. Он позволяет увеличить

значения дат на указанный вами временной интервал. Например, если у вас в ячейке стоит 1 января, то при помощи этого метода можно сгенерировать первое число любого другого месяца.

- `Delete()` — удаляет данные текущего диапазона. С помощью необязательного параметра можно определить, с какой стороны будут сдвигаться ячейки на место удаленных.
- `Dirty()` — помечает ячейки диапазона как "грязные". Такие ячейки будут пересчитаны при следующем же пересчете. Этот метод обычно используется, когда Excel сам не может догадаться, что их нужно пересчитать. Также пересчитать ячейки диапазона можно и принудительно при помощи метода `Calculate()`.
- Методы с префиксом `Fill...` (`FillDown()`, `FillUp()`, `FillLeft()`, `FillRight()`) — позволяют размножить одно и то же значение по ячейкам диапазона в указанном вами направлении.
- `Find()` — позволяет произвести поиск по ячейкам диапазона и вернуть новый объект `Range`, который представляет собой первую ячейку, в которой было найдено нужное значение. У этого метода есть множество необязательных параметров, которые позволяют определить направление поиска, чувствительность к регистру, искать ли значение ячейки целиком или как часть и т. п. Методы `FindNext()` и `FindPrevious()` позволяют продолжить поиск, начатый методом `Find()`, в разных направлениях.
- `GoalSeek()` — позволяет применить автоподбор значений для функции Excel программным способом. На графическом экране то же самое можно сделать при помощи меню **Сервис | Подбор параметра**.
- `Insert()` — позволяет вставить ячейки в диапазон, сдвинув остальные вправо или вниз.
- `Justify()` — позволяет равномерно распределить текст по диапазону. Если в данный диапазон текст не помещается, то он будет распространен на соседние ячейки (с перезаписью их значений).
- `Merge()` — позволяет слить все ячейки диапазона в одну. При этом останется только одно значение — верхней левой ячейки. Разбить обратно такую слитую ячейку на несколько обычных можно при помощи метода `UnMerge()`.
- `Parse()` — позволяет разбить одну ячейку на несколько по указанному вами шаблону (например, чтобы отделить код города от номера телефона).
- `PasteSpecial()` — операция, дополняющая `Copy()` и `Cut()`. Она позволяет вставить то, что лежит в буфере обмена, с указанием специальных параметров вставки (вставлять с добавлением к существующим данным, с умножением, вычитанием, делением и т. п.).

- `PrintOut()` и `PrintPreview()` — позволяют вывести диапазон на печать или открыть режим просмотра перед печатью.
- `Replace()` — метод, дополняющий метод `Find()`. Позволяет проводить поиск и замену значений в диапазоне.
- `Select()` — выделяет указанный диапазон. Объекта `Selection` в Excel нет, вместо него есть возможность получить объект `Range`, представляющий выделенную область.
- `Show()` — экран будет пролистан таким образом, чтобы показать весь указанный диапазон.
- `ShowDependents()` — позволяет пометить стрелками те ячейки, которые зависят от указанного диапазона (только первый уровень зависимости) или убрать эти стрелки. Обратный метод — `ShowPrecedents()`.
- `ShowErrors()` — показывает источник ошибки для указанной ячейки.
- `Sort()` — производит сортировку ячеек в диапазоне. Можно использовать большое количество необязательных параметров для настройки сортировки. `SortSpecial()` — сортировка с учетом особенностей азиатских языков.
- `Speak()` — удивительный метод, который позволяет прочитать вслух содержимое диапазона (можно определить, в каком направлении и будут ли зачитываться формулы). К сожалению, в локализованной версии Excel не работает.
- `SpecialCells()` — очень удобный метод, который позволяет вернуть объект `Range`, включающий в себя все ячейки определенного типа (пустые, с ошибками, с комментариями, последние, с константами, с формулами, с определенным форматированием) и с определенным значением. Например, чтобы вернуть объект `Range`, состоящий из всех пустых ячеек диапазона, можно использовать код:

```
Set oRange2 = oRange.SpecialCells(xlCellTypeBlanks)
oRange2.Select    'Проверяем, так ли это
```

- `SubTotal()` — позволяет посчитать итоговое значение для диапазона (можно выбрать агрегатную функцию и множество других параметров).
- `Table()` — позволяет создать таблицу на основе передаваемого столбца, строки и функции, которую нужно использовать для вычисления ячеек таблицы. Пример в документации по этому методу позволяет автоматически сгенерировать таблицу умножения.
- `TextToColumns()` — сложный метод, который позволяет создать из значения одного столбца несколько столбцов в соответствии с определенным алгоритмом. Принимает множество необязательных параметров.

11.7. Коллекция *QueryTables* и объект *QueryTable*

Для большинства практических задач вполне хватает возможностей объектов `Application`, `Workbook`, `Worksheet` и `Range`. Например, для вставки информации из базы данных вы можете пройти циклом по объекту `ADO.Recordset` и вставить все нужные записи в лист Excel, а затем средствами VBA прописать в нижние строки итоги по вставленным данным. Однако в Excel встроено несколько специальных объектов, которые могут сильно упростить работу в различных ситуациях. Например, ту же операцию по вставке информации из базы данных удобнее будет провести при помощи специального объекта `QueryTable`, который будет рассматриваться в этом разделе. Еще два таких специальных объекта `PivotTable` и `Chart` будут рассматриваться в *разд. 11.8* и *11.9*.

Основное назначение объекта `QueryTable` — работа с набором значений, возвращаемых из базы данных. Этот объект доступен в Excel из графического интерфейса через меню **Данные | Импорт внешних данных | Импортировать данные**. При помощи объектов `QueryTable` вы можете разместить набор записей, полученных с источника данных, на листе Excel для выполнения с ним различных операций (например, анализа при помощи богатой библиотеки функций, для построения диаграмм, отчетов и т. п.). `QueryTable` удобно использовать для "односторонней" работы с источником данных, когда данные только скачиваются с источника в Excel, но изменять данные и сохранять изменения на источнике не нужно. В принципе, в Excel такую возможность синхронизации изменений реализовать можно (например, при помощи перехвата события `Change` объекта `Worksheet`), но намного проще и правильнее будет использовать для этой цели возможности Access. В этом разделе мы будем рассматривать только такую "однонаправленную" передачу данных из базы в Excel.

Как обычно, для того чтобы создать объект `QueryTable` и разместить его на листе, нужно использовать специальную коллекцию `QueryTables`, которая принадлежит рабочему листу (объекту `Worksheet`) и доступна через его одноименное свойство. Свойства и методы объекта `QueryTables` стандартные, как у большинства рассмотренных нами коллекций. Подробного рассмотрения заслуживает только метод `Add()`, при помощи которого и создается объект `QueryTable` (с одновременным добавлением в коллекцию). Этот метод принимает три параметра:

- `Connection` — источник данных для `QueryTable` (в виде объекта типа `Variant`). В качестве источника данных можно использовать:

- строку подключения OLE DB или ODBC *(см. гл. 9)*;
- готовый объект `Recordset`, созданный стандартными средствами ADO или DAO. При этом можно изменять `Recordset`, на который ссылается `QueryTable` и обновлять `QueryTable`. По многим причинам это самый удобный вариант при работе с `QueryTable`;
- другой объект `QueryTable` (вместе со строкой подключения и текстом запроса);
- текстовый файл;
- результаты Web-запроса или запроса Microsoft Query (в виде файла DQY или IQY). Создать такой файл запроса можно при помощи графических средств Excel в меню **Данные | Импорт внешних данных | Создать запрос**.

❒ `Destination` — куда вставлять полученную `QueryTable`. Передается объект `Range`, и вставка производится, начиная с верхнего левого угла диапазона.

❒ `SQL` — при помощи этого необязательного параметра можно определить SQL-запрос, который будет выполняться на источнике данных ODBC. Этот же запрос можно определить при помощи одноименного свойства объекта `QueryTable`.

Конечно, правильнее всего при создании `QueryTable` использовать готовый объект `Recordset`. В этом случае у нас будут и самые полные возможности настройки подключения и курсора, и возможность эффективного промежуточного хранения данных в оперативной памяти (в объекте `Recordset`), куда можно вносить изменения, и все удобные свойства и методы объекта `Recordset`. Код на создание объекта `QueryTable` на листе Excel может выглядеть так (мы используем тот же `Recordset` на основе таблицы `Northwind.Customers`, что и в *разд. 9.5*):

```
Dim cn As ADODB.Connection
Set cn = CreateObject("ADODB.Connection")
cn.Provider = "SQLOLEDB"
cn.ConnectionString = "User ID=SA;Password=password;" _
    & "Data Source = LONDON1;Initial Catalog = Northwind"
cn.Open
Dim rs As ADODB.Recordset
Set rs = CreateObject("ADODB.Recordset")
rs.Open "select * from dbo.customers", cn
Dim QT1 As QueryTable
Set QT1 = QueryTables.Add(rs, Range("A1"))
QT1.Refresh
```

Непосредственно помещение объекта `QueryTable` на лист производится при помощи метода `QueryTable.Refresh()`. Без него объект `QueryTable` будет создан только в оперативной памяти.

Информация о самых важных свойствах и методах объекта `QueryTable` представлена далее.

- `BackgroundQuery` — определяет, может ли выполнение запроса производиться в фоновом режиме, пока пользователь совершает в Excel другие действия. По умолчанию установлено в `True`. В `False` следует переводить только тогда, когда пользователь своими действиями в Excel может как-то помешать нормальной работе приложения.
- `CommandText` — текст команды SQL, т. е. текст запроса, который передается на источник. Существует совместно с аналогичным свойством `SQL` (которое оставлено для обратной совместимости) и имеет перед ним приоритет. При передаче `QueryTable` готового `Recordset` это свойство недоступно.
- `CommandType` — тип передаваемой в `CommandText` команды (вся таблица, SQL-запрос, имя куба OLAP и т. п.). При работе с готовым `Recordset` также недоступно.
- `Connection` — строка подключения, которую можно передать при вызове метода `Add()` коллекции `QueryTables`. Также при работе с готовым `Recordset` недоступно.
- `Destination` — второй параметр, который передается методу `Add()`. Возвращает объект `Range`, представляющий верхнюю левую ячейку диапазона, занимаемого на листе объектом `QueryTable`. После создания `QueryTable` доступен только для чтения.
- `EnableEditing` — определяет, может ли пользователь изменять на графическом экране свойства объекта `QueryTable`. По умолчанию установлено в `True`. Если перевести в `False`, то пользователь сможет только обновлять `QueryTable`.
- `EnableRefresh` — определяет, может ли пользователь обновлять `QueryTable`, получая заново данные (с источника или из `Recordset`).
- `FetchedRowOverflow` — это свойство принимает значение `True`, если записи, полученные с источника, не поместились на листе Excel (т. е. было скачано больше, чем 65 536 записей). Ошибки в такой ситуации не возникает, поэтому если вы работаете с большими наборами записей, имеет смысл реализовать соответствующие проверки.
- `FieldNames` — очень полезное свойство. Позволяет отключить вставку полученных с источника названий столбцов в первую строку `QueryTable`. По умолчанию `True` — вставлять названия столбцов.

- `MaintainConnection` — определяет, будет ли соединение с источником открыто все время до закрытия листа. По умолчанию `True` — оптимизировано для выполнения частых обновлений. Если переставить в `False`, можно сэкономить оперативную память на клиенте за счет уменьшения скорости обновления данных.
- `Name` — имя объекта `QueryTable` (на графическом экране его можно посмотреть, если на панели управления **Внешние данные** нажать кнопку **Свойства диапазона данных**). По умолчанию задается как `ExternalData_` *номер*.
- `Parameters` — позволяет получить доступ к коллекции `Parameters` (набор параметров запроса). Возможности практически такие же, как у параметров объекта `Recordset`.
- `PreserveColumnInfo` и `PreserveFormatting` — определяют, сохранять ли информацию о столбцах (сортировке, фильтрации и т. п.) и форматировании после обновления `QueryTable`. По умолчанию все сохраняется.
- `QueryType` — позволяет выяснить, что использовалось при создании `QueryTable`: `Recordset`, прямой доступ к таблице, SQL-запрос и т. п. Свойство доступно только для чтения.
- `Recordset` — возвращает ссылку на объект `Recordset`, который использовался для создания `QueryTable`, или позволяет заменить его для объекта `QueryTable` на другой (изменения вступят в силу только после вызова метода `Refresh()`).
- `Refreshing` — это свойство принимает значение `True` на момент выполнения фонового запроса к источнику. Если выполнение запроса слишком затянулось, его можно прервать при помощи метода `CancelRefresh()`.
- `RefreshOnFileOpen` — определяет, обновлять ли данные автоматически при открытии листа или можно обойтись уже скачанными значениями (по умолчанию).
- `RefreshPeriod` — определяет, через какие интервалы времени автоматически будет обновляться информация в `QueryTable` данными с источника. По умолчанию задан 0, т. е. автоматическое обновление отключено.
- `RefreshStyle` — определяет, что делать с существующими ячейками, на место которых вставляются ячейки `QueryTable` при обновлении.
- `ResultRange` — это, наверное, самое важное свойство объекта `QueryTable`. Как правило, данные из базы перекачиваются в Excel для дальнейшей обработки. Это свойство позволяет получить диапазон, который включает в себя все ячейки, вставленные на лист из объекта `QueryTable`, чтобы потом применить к ним различные функции (обычно по столбцам или по строкам). Чтобы этот метод сработал, обязательно нужно провести вставку

данных QueryTable на лист при помощи метода Refresh(). После этого можно будет использовать то, что возвращает это свойство, как обычный диапазон. Самый простой способ продемонстрировать работу этого метода — воспользоваться кодом:

```
QT1.ResultRange.Select
```

А следующий пример генерирует под первым столбцом QueryTable формулу с суммированием значений этого столбца:

```
Set c1 = Sheets("Лист1").QueryTables(1).ResultRange.Columns(1)
c1.Name = "Column1"
c1.End(xlDown).Offset(1, 0).Formula = "=SUM(Column1)"
```

- RowNumbers — свойство, которое может сильно упростить работу с данными, полученными при помощи QueryTable. Позволяет сгенерировать еще один столбец в QueryTable (слева), который будет состоять из номеров записей, полученных через QueryTable.
- SaveData — определяет, сохранять ли данные, полученные через QueryTable, вместе с книгой Excel. По умолчанию задано True. В False имеет смысл переводить это свойство для того, чтобы изначально гарантировать работу пользователя только с самыми последними данными, полученными из источника.
- SavePassword — определяет, сохранять ли пароль вместе со строкой подключения (это свойство можно использовать только для источников ODBC). Если переставить его в False, то можно повысить уровень безопасности вашего приложения.
- SourceDataFile — полный путь и имя файла источника (для Access, DBF и других настольных СУБД). Для клиент-серверных систем (таких как SQL Server) возвращает Null.
- Свойства с префиксом Text... — определяют параметры текстового файла, если этот файл был выбран в качестве источника для QueryTable.
- Свойства с префиксом Web... — определяют параметры данных, получаемых от запроса к Web-источнику.

Методы Refresh(), CancelRefresh() и Delete() объекта QueryTable очевидны и каких-либо комментариев не требуют. Метод ResetTimer() позволяет обнулить таймер автоматического обновления, а метод SaveAsODC() — сохранить определение источника данных в виде файла Microsoft Query (если источником был объект Recordset, то этот метод вернет ошибку).

У объекта QueryTable есть также два события: BeforeRefresh и AfterRefresh. Они срабатывают соответственно перед началом загрузки данных с источника и после окончания загрузки.

11.8. Работа со сводными таблицами (объект *PivotTable*)

В процессе работы большинства предприятий накапливаются так называемые необработанные данные *(raw data)* о деятельности. Например, для торгового предприятия могут накапливаться данные о продажах товаров (по каждой покупке отдельно), для предприятий сотовой связи — статистика нагрузки на базовые станции и т. п. Очень часто менеджменту предприятия необходима аналитическая информация, которая генерируется на основе необработанных данных, например, посчитать вклад каждого вида товара в доходы предприятия или качество обслуживания в зоне данной станции. Из необработанной информации такие сведения извлечь очень тяжело: нужно выполнять сложные SQL-запросы, которые обрабатываются долго и часто мешают текущей работе. Поэтому все чаще в настоящее время необработанные данные сводятся вначале в хранилище архивных данных Data Warehouse, а затем — в кубы OLAP, которые очень удобны для интерактивного анализа. Проще всего представить себе кубы OLAP как многомерные таблицы, в которых вместо стандартных двух измерений (столбцы и строки, как в обычных таблицах) измерений может быть очень много. Обычно для описания измерений в кубе используется термин "в разрезе". Например, отделу маркетинга может быть нужна информация во временном разрезе, в региональном разрезе, в разрезе типов продукта, в разрезе каналов продаж и т. п. При помощи кубов (в отличие от стандартных SQL-запросов) очень просто получать ответы на такие вопросы, как "сколько товаров такого-то типа было продано в четвертом квартале прошлого года в Северо-Западном регионе через региональных дистрибьюторов".

Конечно, в обычных базах данных такие кубы не создать. Для работы с кубами OLAP требуются специализированные программные продукты. Вместе с SQL Server поставляется база данных OLAP от Microsoft, которая называется Analysis Services. Есть OLAP-решения от Oracle, IBM, Sybase и других фирм.

Для работы с такими кубами в Excel встроен специальный клиент. По-русски он называется **Сводная таблица** (на графическом экране он доступен через меню **Данные | Сводная таблица**), а по-английски — **Pivot Table**. Соответственно, объект, который представляет этот клиент, называется `PivotTable`. Необходимо отметить, что он умеет работать не только с кубами OLAP, но и с обычными данными в таблицах Excel или в базах данных, но многие возможности при этом теряются.

Сводная таблица и объект `PivotTable` — это программные продукты фирмы Panorama Software, которые были приобретены Microsoft и интегрированы в Excel. Поэтому работа с объектом `PivotTable` несколько отличается от работы

с другими объектами Excel. Догадаться, что нужно сделать, часто бывает непросто. Поэтому рекомендуется для получения подсказок активно использовать макрорекордер. В то же время при работе со сводными таблицами пользователям часто приходится выполнять одни и те же повторяющиеся операции, поэтому автоматизация во многих ситуациях необходима.

При работе со сводной таблицей первое, что нам потребуется сделать, — это создать объект `PivotCache`, который будет представлять собой набор записей, полученных с источника OLAP. Очень условно этот объект `PivotCache` можно сравнить с `QueryTable`. Для каждого объекта `PivotTable` можно использовать только один объект `PivotCache`. Создание объекта `PivotCache` производится при помощи метода `Add()` коллекции `PivotCaches`:

```
Dim PC1 As PivotCache
Set PC1 = ActiveWorkbook.PivotCaches.Add(xlExternal)
```

`PivotCaches` — стандартная коллекция, из ее методов, которые заслуживают подробного рассмотрения, можно назвать только метод `Add()`. Этот метод принимает два параметра:

- `SourceType` — обязательный, определяет тип источника данных для сводной таблицы. Можно указать создание `PivotTable` на основе диапазона в Excel, данных из базы данных, во внешнем источнике данных, другой `PivotTable` и т. п. На практике OLAP имеет смысл использовать только тогда, когда данных много — нужно специализированное внешнее хранилище (например, Microsoft Analysis Services). В этой ситуации выбирается значение `xlExternal`.
- `SourceData` — обязательный во всех случаях, кроме тех, когда значение первого параметра равно `xlExternal`. Определяет тот диапазон данных, на основе которого и будет создаваться `PivotTable`. Обычно принимает в качестве значения объект `Range`.

Следующая задача — это настроить параметры объекта `PivotCache`. Как уже говорилось, этот объект очень напоминает `QueryTable`, и набор свойств и методов у него очень похожий. Некоторые наиболее важные свойства и методы представлены далее.

- `ADOConnection` — возвращает объект `ADO.Connection`, который автоматически создается при подключении к внешнему источнику данных. Используется для дополнительной настройки свойств подключения.
- `Connection` — работает точно так же, как и одноименное свойство объекта `QueryTable`. Может принимать строку подключения, готовый объект `Recordset`, текстовый файл, Web-запрос, файл Microsoft Query. Чаще всего при работе с OLAP строка подключения прописывается напрямую (по-

скольку получать объект Recordset, например, для изменения данных смысла нет — источники данных OLAP практически всегда доступны только для чтения). Настройка этого свойства для подключения к базе данных FoodMart (учебная база данных Analysis Services) на сервере LONDON может выглядеть так:

```
PC1.Connection = "OLEDB;Provider=MSOLAP.2;Data Source=LONDON1;" _
                 & "Initial Catalog = FoodMart 2000"
```

- CommandType и CommandText — описывают тип команды, которая передается на сервер баз данных, и текст самой команды. Например, чтобы обратиться к кубу Sales и получить его целиком в кэш на клиенте, можно использовать код типа:

```
PC1.CommandType = xlCmdCube
PC1.CommandText = Array("Sales")
```

- LocalConnection — позволяет подключиться к локальному кубу (файлу cub), созданному средствами Excel. Конечно, такие файлы для работы с "производственными" объемами данных использовать очень не рекомендуется — только для целей создания макетов и т. п.
- MemoryUsed — возвращает количество оперативной памяти, используемой PivotCache. Если PivotTable на основе этого PivotCache еще не создана и не открыта, то возвращается 0. Можно использовать для проверки, если ваше приложение будет работать на слабых клиентах.
- OLAP — возвращает True, если PivotCache подключен к серверу OLAP.
- OptimizeCache — позволяет оптимизировать структуру кэша. Изначальная загрузка данных будет производиться дольше, но потом скорость работы может существенно возрасти. Для источников OLE DB эта оптимизация не работает.

Остальные свойства объекта PivotCache совпадают с аналогичными свойствами объекта QueryTable, и поэтому здесь рассматриваться не будут.

Главный метод объекта PivotCache — это метод CreatePivotTable(). С его помощью и производится следующий этап — создание сводной таблицы (объекта PivotTable). Этот метод принимает четыре параметра:

- TableDestination — единственный обязательный параметр. Принимает объект Range, в верхний левый угол которого будет помещена сводная таблица.
- TableName — имя сводной таблицы. Если не указано, то автоматически сгенерируется имя типа "СводнаяТаблица1".

- `ReadData` — если установить в `True`, то все содержимое куба будет автоматически помещено в кэш. С этим параметром нужно быть очень осторожным, поскольку неправильное его применение может резко увеличить нагрузку на клиента.
- `DefaultVersion` — это свойство обычно не указывается. Позволяет определить версию создаваемой сводной таблицы. По умолчанию задается наиболее свежая версия.

Создание сводной таблицы в первой ячейке первого листа книги может выглядеть так:

```
PC1.CreatePivotTable Range("A1")
```

Сводная таблица у нас создана, однако сразу после создания она пуста. В ней предусмотрено четыре области, в которые можно размещать поля из источника:

- *область столбцов* — в нее помещаются те измерения ("разрез", в котором будут анализироваться данные), записей в которых меньше;
- *область строк* — те измерения, записей в которых больше;
- *область страницы* — те измерения, по которым нужно только проводить фильтрацию (например, показать данные только по такому-то региону или только за такой-то год);
- *область данных* — центральная часть таблицы. Те числовые данные (например, сумма продаж), которые мы будем анализировать.

Полагаться на пользователя в том, что он правильно разместит элементы во всех четырех областях, не стоит. Кроме того, это может занять определенное время. Поэтому часто требуется расположить данные в сводной таблице программным образом. Эта операция производится при помощи объекта `CubeField`. Главное свойство этого объекта — `Orientation`, оно определяет, где будет находиться то или иное поле. Например, помещаем измерение `Customers` в область столбцов:

```
PT1.CubeFields("[Customers]").Orientation = xlColumnField
```

измерение `Time` — в область строк:

```
PT1.CubeFields("[Time]").Orientation = xlRowField
```

измерение `Product` — в область страницы:

```
PT1.CubeFields("[Product]").Orientation = xlPageField
```

и, наконец, показатель `Unit Sales` (числовые данные для анализа):

```
PT1.CubeFields("[Measures].[Unit Sales]").Orientation = xlDataField
```

Теперь сводная таблица создана, и с ней можно работать. Однако часто необходимо выполнить еще одну операцию — раскрыть нужный уровень иерархии измерения. Например, если нас интересует поквартальный анализ, то нужно раскрыть уровень `Quarter` измерения `Time` (по умолчанию показывается только самый верхний уровень). Конечно, пользователь может сделать это самостоятельно, но не всегда можно рассчитывать, что он догадается, куда щелкнуть мышью. Программным образом раскрыть иерархию измерения `Time` на уровень кварталов для 1997 г. можно при помощи объектов `PivotField` и `PivotItem`:

```
PT1.PivotFields("[Time].[Year]").PivotItems("[Time].[1997]").
DrilledDown = True
```

11.9. Работа с диаграммами (объект *Chart*)

Одно из основных применений Excel — это анализ данных. А для анализа данных часто удобно использовать диаграммы с их специальными возможностями, такими как тренды. На практике задачи по автоматизации создания множества похожих друг на друга диаграмм (обычно на основе информации, полученной из базы данных) возникают очень часто.

С диаграммами в Excel существует некоторая терминологическая путаница. То, что на графическом интерфейсе русского Excel называется диаграммой (меню **Вставка | Диаграмма**), по-английски называется графиком *(Chart)* и ему соответствует объект `Chart`. В объектной модели Excel предусмотрен также и объект `Diagram`, но он представляет скорее схему отношений (то, что при помощи графического интерфейса русского Excel можно добавить при помощи меню **Вставка | Схематическая диаграмма**). Под диаграммой в этом разделе будет пониматься то же, что и у создателей русского Excel — обычный график.

Диаграммы в Excel создаются при помощи объекта `Chart`. Вначале лучше этот объект объявить:

```
Dim oChart As Chart
```

Дальше можно создавать диаграмму. Создание диаграммы производится при помощи много раз использованного нами приема — вызова метода `Add()` коллекции `Charts`:

```
Set oChart = ActiveWorkbook.Charts.Add(, ActiveSheet)
```

В принципе, диаграмма уже создана, но поскольку никакие ее свойства не определены, она выглядит как пустой лист. Чтобы она обрела содержание, необходимо выполнить еще несколько действий.

Первое (и единственное обязательное действие) — определить источник данных для диаграммы, для чего предназначен метод `SetSourceData()`. В качестве источника может выступать только объект `Range` (он передается в качестве первого и единственного обязательного параметра этого метода). Второй параметр (необязательный) определяет, в каком порядке считывать данные — сначала по столбцам, потом по строкам или наоборот. Например, в нашем случае это может выглядеть так:

```
oChart.SetSourceData Sheets("Лист1").Range("A1:A10")
```

В принципе, если запустить этот код на выполнение, то диаграмма уже будет создана. Для всех остальных параметров будут приняты значения по умолчанию. Однако на практике нужно определить еще хотя бы тип диаграммы (по умолчанию она будет выглядеть как "обычная гистограмма", т. е. ряд из столбиков разной длины). Для этой цели используется свойство `ChartType`, для которого разработчиками предусмотрено 73 значения. Например, чтобы преобразовать диаграмму в обычный график, можно использовать код вида:

```
oChart.ChartType = xlLineMarkers
```

Еще одна очень распространенная задача — добавить дополнительные ряды на диаграмму. Для этой цели необходимо создать и получить ссылку на объект `Series` — ряд, а потом для ряда определить свойство `Values` (ему передается в качестве значения объект `Range`):

```
Dim oSeries As Series
Set oSeries = oChart.SeriesCollection.NewSeries
oSeries.Values = Worksheets(1).Range("B1:B10")
```

Пользователи часто говорят, что им необходимо создавать диаграммы не на отдельном листе, а на том же листе, на котором расположены данные. По умолчанию диаграмма создается в оперативной памяти и помещается на отдельный лист. Если нам необходимо поместить ее на уже существующий лист, то в этом случае ее надо создать вначале на отдельном листе, а затем переместить при помощи метода `Location`. Отдельный лист, созданный для диаграммы, при этом автоматически исчезнет:

```
oChart.Location xlLocationAsObject, "Лист1"
```

Обратите внимание, что метод `Location` принимает в качестве первого параметра одну из констант (`xlLocationAsNewSheet` — переместить на специально создаваемый новый лист, `xlLocationAsObject` — переместить на объект, т. е. на лист), а в качестве второго — не объект листа, а его имя. Если код предполагается использовать и в русской, и в английской версии Excel, то предпочтительнее получать имя листа программным образом.

Большая неприятность, связанная с методом `Location`, заключается в том, что после перемещения диаграммы внутрь листа объектная ссылка на эту диа-

грамму теряется, и надо находить объект этой диаграммы заново. При попытке повторного обращения к объекту `Chart` выдается сообщение `"Automation Error"`. Поэтому лучше всего вызов метода `Location` помещать в самый конец кода, посвященного диаграмме. В противном случае нам придется разыскивать созданную нами диаграмму и заново получать на нее объектную ссылку, например, так:

```
Dim oSeries As Series
Set oSeries = _
    Worksheets(1).ChartObjects(1).Chart.SeriesCollection.NewSeries
oSeries.Values = Worksheets(1).Range("B1:B10")
```

Так работать, конечно, очень неудобно.

Остальные многочисленные параметры диаграммы настраиваются при помощи свойств и методов объектов `Chart`.

- `ChartArea` — это свойство возвращает одноименный объект `ChartArea`, который представляет собой область, занимаемую диаграммой, и используется для настройки внешнего вида диаграммы (свойства `Font`, `Interior` и т. п.). Если необходимо настроить внешний вид не всей диаграммы, а той ее части, которая используется непосредственно для вывода графика, используется схожее свойство `PlotArea`. По умолчанию диаграмма размещается прямо по центру листа. Если необходимо ее переместить в точно определенное место листа, используются знакомые свойства `Top`, `Height`, `Left` и `Width` объекта `ChartArea`.
- `ChartTitle` — возвращает одноименный объект, при помощи которого можно настроить заголовок диаграммы (с такими свойствами, как `Text`, `Font`, `Border` и т. п.).
- `ChartType` — важнейшее свойство, про которое мы уже говорили. Определяет тип диаграммы.
- `HasDataTable` — если установить это свойство в `True`, то в нижней части диаграммы (по умолчанию) появится таблица с числами, на основе которых была создана диаграмма. Одновременно будет создан программный объект `DataTable`, при помощи которого можно будет настроить представление этой таблицы. Схожим образом работают свойства `HasLegend`, `HasPivotFields` и `HasTitle`.
- `Name` — это свойство позволяет настроить имя диаграммы (как название вкладки в Excel). По умолчанию диаграммы называются последовательно `"Диаграмма1"`, `"Диаграмма2"` и т. п.
- `SizeWithWindow` — если поставить значение этого свойства в `True`, то размер диаграммы будет подогнан таким образом, чтобы точно соответствовать размеру листа. По умолчанию установлено в `False`.

- `Tab` — свойство, о котором мало кто знает. Оно позволяет настроить при помощи одноименного объекта внешний вид вкладки в книге Excel для диаграммы (или просто листа). Например, чтобы пометить вкладку зеленым цветом, можно воспользоваться кодом:

```
oChart.Tab.Color = RGB(0, 255, 0)
```

- `Visible` — позволяет спрятать диаграмму без ее удаления.

Остальные свойства в основном относятся к настройке отображения трехмерных диаграмм и к защите диаграммы от изменения пользователем.

Далее представлены самые важные методы объекта `Chart`.

- `Activate()` — используется очень часто. Он позволяет сделать диаграмму активной (т. е. просто перейти на нее).
- `ApplyCustomType()` — позволяет создать диаграмму своего собственного пользовательского типа (для этого необходимо вначале создать шаблон для этого типа и поместить его в галерею).
- `ApplyDataLabels()` — позволяет поместить на диаграмму метки для размещенных на ней данных. Этот метод принимает множество параметров, которые позволяют настроить отображение меток (показывать или не показывать значения и т. п.).
- `Axes()` — возвращает объект, представляющий оси диаграммы. Затем этот объект можно использовать для настройки данных осей.
- `ChartWizard()` — этот метод позволяет быстро переформатировать диаграмму, как если бы вы прошли на графическом экране Мастер построения диаграмм и передали ему значения. Позволяет при помощи одной строки кода добиться того, что другими способами заняло бы несколько строк.
- `Copy()` — позволяет скопировать диаграмму в другое место книги (например, для создания новой диаграммы на основе существующей). Для переноса существующей диаграммы в другое место можно воспользоваться методами `Location()` или `Move()`.
- `CopyPicture()` — замечательный метод, который позволяет поместить диаграмму в буфер обмена как изображение. Затем это изображение можно вставить, например, в документ Word или в любое другое место. Еще один вариант — воспользоваться методом `Export()`, который позволяет создать рисунок, представляющий диаграмму, в виде файла на диске.
- `Delete()` — удаляет диаграмму.
- `Evaluate()` — как обычно, этот метод позволяет найти нужную диаграмму в книге по ее имени.

- `PrintOut()` — отправляет диаграмму на печать. Этот метод принимает множество параметров, которые позволяют настроить такой вывод.
- `Refresh()` — позволяет обновить диаграмму, если изменились данные, на основе которых она строилась.
- `Select()` — выделяет диаграмму (равносильно щелчку по ней левой кнопкой мыши). Обратный метод `Deselect()` снимет выделение (равносильно нажатию <Esc>).
- `SetBackgroundPicture()` — позволяет "подложить" под диаграмму фоновый рисунок. Конечно, он должен быть не очень ярким.
- `SetSourceData()` — важнейший метод, который позволяет определить данные, на основе которых строится диаграмма. Про него мы уже говорили.

Для объекта `Chart` предусмотрено также события "на все случаи жизни" — реакции на щелчки мышью, на выделение или снятие выделения, активизацию, пересчет данных, изменение размера и т. п., однако на практике такие события используются редко.

11.10. Другие объекты Excel

В Excel предусмотрено множество других объектов, подробно рассмотреть которые не позволяет объем данной книги. Далее представлен обзор самых важных из них. Найти более подробную информацию о работе с ними можно при помощи официальной документации или макрорекордера.

Объект `CellFormat` позволяет настраивать форматирование ячеек Excel — шрифт, рамки, цветовое оформление и т. п. При этом можно использовать этот объект для копирования оформления с других ячеек, для поиска и замены по оформлению и т. п. Условное форматирование можно настроить при помощи объекта `FormatCondition`.

`ListObject` — новый объект (появился только в Office 2003), который предназначен для работы со списками — наборами взаимосвязанных данных (например, списки сотрудников). Обычно используется при работе с книгами Excel на SharePoint Portal Server.

Объект `Validation` позволяет настроить проверку вводимых пользователем данных.

Объект `Watch` позволяет настроить контрольное значение для формул. При помощи контрольного значения (меню **Сервис | Макрос | Показать контрольное значение**) можно отслеживать значения тех формул, которые находятся за пределами экрана.

Задание для самостоятельной работы 11: Применение Excel для анализа информации из базы данных

ЗАДАНИЕ:

В вашей компании ведется учет товаров, которые имеются на складе, при помощи таблицы `Товары` базы данных `Борей`, которая расположена в каталоге C:\Program Files\Microsoft Office\OFFICE11\SAMPLES. В этой таблице находятся следующие важные для вас столбцы:

- `КодТовара` — идентификатор товара;
- `Марка` — наименование продукта;
- `Цена` — стоимость за единицу продукта;
- `НаСкладе` — количество единиц этого товара на складе;
- `МинимальныйЗапас` — минимально допустимое количество единиц данного товара на складе. Если реальное количество единиц этого товара меньше, чем это значение, то товар нужно срочно заказать;
- `ПоставкиПрекращены` — флаг прекращения работы с товаром. Если в этом столбце стоит единица, то это значит, что принято решение закупки этого товара больше не производить.

Все остальные столбцы для целей этой работы можно игнорировать.

Заполнение таблицы `Товары` производится при помощи специализированного приложения, созданного достаточно давно и не предусматривающего некоторых необходимых форм.

Вам поручено создать приложение на основе Excel, которое бы:

1. Производило вставку в лист Excel данных по всем строкам и всем столбцам этой таблицы.
2. Генерировало бы в Excel дополнительные столбцы следующего содержания:
   - `"Заказать товара, штук"` — разница между столбцами `МинимальныйЗапас` и `НаСкладе`. В этот столбец должна помещаться информация о количестве товара в штуках, которое нужно срочно заказать. Эту информацию нужно генерировать только для тех записей, для которых значение в столбце `МинимальныйЗапас` больше, чем в столбце `НаСкладе`, и у которых значение столбца `ПоставкиПрекращены` установлено в `False`;
   - `"Стоимость заказа"` — определяло бы стоимость такого пополнения склада для каждой строки в таблице. Стоимость заказа рассчитывается как произведение предыдущего столбца и столбца `Цена`. Эту информа-

цию также нужно генерировать только для тех записей, для которых значение в столбце МинимальныйЗапас больше, чем в столбце НаСкладе.

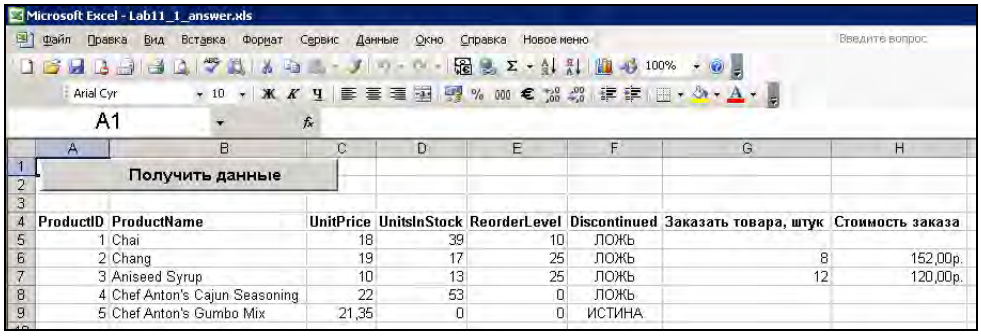
Примечание

В реальной задаче правильнее (и намного производительнее) бы было перенести расчет таких столбцов на сервер баз данных, использовав SQL-запрос с вычисляемыми столбцами. Однако для целей этой самостоятельной работы реализуйте их вставку средствами Excel (такое решение — единственно возможное, к примеру, если мы обращаемся к нереляционному источнику данных, такому как текстовый файл).

3. Вставляло бы в одной строке под полученными записями из базы данных две итоговые строки:
   - "Общая стоимость товаров на складе" — итоговая стоимость всех товаров, которые находятся на складе (как сумма произведений столбцов НаСкладе и Цена для каждой строки);
   - "Общая стоимость товаров к заказу" — итог по столбцу СтоимостьЗаказа.

Общий вид получившегося приложения может быть таким, как представлено на рис. 11.2.

Итоговые строки могут выглядеть так, как показано на рис. 11.3.



	ProductID	ProductName	UnitPrice	UnitsInStock	ReorderLevel	Discontinued	Заказать товара, штук	Стоимость заказа
5	1	Chai	18	39	10	ЛОЖЬ		
6	2	Chang	19	17	25	ЛОЖЬ	8	152,00р.
7	3	Aniseed Syrup	10	13	25	ЛОЖЬ	12	120,00р.
8	4	Chef Anton's Cajun Seasoning	22	53	0	ЛОЖЬ		
9	5	Chef Anton's Gumbo Mix	21,35	0	0	ИСТИНА		

Рис. 11.2. Первые строки листа с импортированными данными

73	69	Gudbrandsdalsost	36	26	15	ЛОЖЬ		
74	70	Outback Lager	15	15	30	ЛОЖЬ	15	225,00р.
75	71	Flotemysost	21,5	26	0	ЛОЖЬ		
76	72	Mozzarella di Giovanni	34,8	14	0	ЛОЖЬ		
77	73	Röd Kaviar	15	101	5	ЛОЖЬ		
78	74	Longlife Tofu	10	4	5	ЛОЖЬ	1	10,00р.
79	75	Rhönbräu Klosterbier	7,75	125	25	ЛОЖЬ		
80	76	Lakkalikööri	18	57	20	ЛОЖЬ		
81	77	Original Frankfurter grüne Soße	13	32	15	ЛОЖЬ		
82								
83								
84		**Общая стоимость товаров на складе:**	**74 050,85р.**					
85		**Общая стоимость товаров к заказу:**	**3 633,45р.**					
86								

Рис. 11.3. Последние строки с итоговыми значениями

Ответ к заданию 11

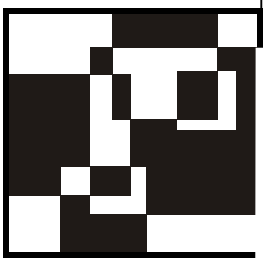
1. Создайте новый файл Excel, сделайте видимым панель управления **Элементы управления**, щелкните в нем по элементу управления **Кнопка** и поместите кнопку на лист Excel. Для наших целей мы будем считать, что созданная кнопка занимает две верхние строки первого листа.
2. На панели инструментов **Элементы управления** щелкните по кнопке **Свойства** (при этом созданная кнопка должна быть выделена) и настройте для свойства `Caption` значение `"Получить данные"`. Воспользуйтесь свойством `Font`, чтобы настроить подходящий шрифт для вашей кнопки.
3. Щелкните правой кнопкой мыши по созданной вами кнопке и в контекстном меню выберите **Исходный текст**. Откроется редактор Visual Basic с курсором ввода на месте события `Click` для вашей кнопки.
4. В окне редактора кода в меню **Tools** выберите **References** и установите флажок напротив строки **Microsoft ActiveX Data Objects 2.1 Library**.
5. Код для события `Click` вашей кнопки **Получить данные** может быть таким, как показано далее.

```
Private Sub CommandButton1_Click()
    'Вначале — чистим всю книгу от старых данных
    Cells.Select
    Selection.Clear

    'Создаем и настраиваем объект Connection
    Dim cn As New ADODB.Connection
    cn.ConnectionString = "Provider=Microsoft.Jet.OLEDB.4.0;" _
& "Data Source=C:\Program Files\Microsoft Office\OFFICE11\SAMPLES\Борей.mdb"
    cn.Open

    'Создаем и настраиваем объект Recordset
    Dim rs As New ADODB.Recordset
    rs.Open "SELECT [КодТовара], [Марка], [Цена], [НаСкладе]," _
        & "[МинимальныйЗапас], [ПоставкиПрекращены] FROM Товары", cn

    'На основе Recordset создаем объект QueryTable и
    'вставляем его, начиная с 4-й строки
    Dim QT1 As QueryTable
    Set QT1 = QueryTables.Add(rs, Range("A4"))
    QT1.Refresh

    'Определяем количество записей в QueryTable
    Dim nRowCount As Integer
    Dim oRange As Range
```

```
Set oRange = QT1.ResultRange
nRowCount = oRange.Rows.Count

'Формируем столбец "Заказать товара, штук"
Range("G4").Value = "Заказать товара, штук"
Range("G4").Font.Bold = True
Range("G4").Columns.AutoFit

'Формируем столбец "Стоимость заказа"
Range("H4").Value = "Стоимость заказа"
Range("H4").Font.Bold = True
Range("H4").Columns.AutoFit

'Создаем диапазон, который включит в себя столбец G
' "вдоль" QueryTable
Set oRange = Range("G5", "G" & nRowCount + 3)

'Готовим переменные, которые нам потребуются в цикле
Dim oCell As Range
Dim sRowNumber As String
Dim cMoney As Currency
Dim cItogMoney As Currency
Dim cItogSklad As Currency

'Проходим циклом по всем ячейкам созданного диапазона
For Each oCell In oRange.Cells
    'Получаем абсолютный номер строки в виде строковой переменной
    sRowNumber = Replace(oCell.Address(True), "$G$", "")
    'Проверяем определенные нами условия
    If Range("E" & sRowNumber).Value > Range("D" & sRowNumber) And _
        Range("F" & sRowNumber).Value = False Then

        'Получаем значение для столбца G (заказ в штуках)
        oCell.Value = (CInt(Range("E" & sRowNumber).Value) - _
                       CInt(Range("D" & sRowNumber).Value))

        'Получаем значение для столбца H (стоимость заказа)
        cMoney = (CInt(Range("E" & sRowNumber).Value) - _
                  CInt(Range("D" & sRowNumber).Value)) * _
                  CCur(Range("C" & sRowNumber).Value)
        'Записываем его в столбец H
        Range("H" & sRowNumber).Value = cMoney
        'Сразу плюсуем к итогу в рублях
        cItogMoney = cItogMoney + cMoney
    End If
```

```
        'И в том же цикле сразу суммируем стоимость товаров на складе
        cItogSklad = cItogSklad + (Range("C" & sRowNumber).Value * _
                    Range("D" & sRowNumber).Value)

    Next

    'Формируем две строки с итогами
Range("B" & nRowCount + 6).Value = "Общая стоимость товаров на складе:"
Range("B" & nRowCount + 6).Font.Bold = True
Range("B" & nRowCount + 7).Value = "Общая стоимость товаров к заказу:"
    Range("B" & nRowCount + 7).Font.Bold = True
    Range("D" & nRowCount + 6).Value = cItogSklad
    Range("D" & nRowCount + 6).Font.Bold = True
    Range("D" & nRowCount + 7).Value = cItogMoney
    Range("D" & nRowCount + 7).Font.Bold = True

    'Для красоты выделяем итоговое значение ...
    Range("D" & nRowCount + 7).Select
    '... и производим скроллирование
    Range("D" & nRowCount + 7).Show

End Sub
```

ГЛАВА 12



Программирование в Access

12.1. Отличительные особенности создания приложений в Access

Программирование в Access сильно отличается от программирования в Word, Excel и других приложениях Office. Главное принципиальное отличие заключается в том, что Word, Excel, PowerPoint, Project и т. п. предназначены, прежде всего, для непосредственной работы пользователя, без какой-либо их доработки со стороны разработчиков предприятия. Access иногда используется пользователями как конечное приложение, но чаще он все-таки применяется как платформа для создания своих приложений разработчиками.

Второе отличие заключается в том, что в Access встроено свое собственное ядро для работы с данными. Фактически Access — это полноценная система управления базами данных, поэтому для полного использования его возможностей необходимы знания о принципах работы с базами данных: что такое таблицы и отношения между ними (система ключей), что такое нормализация данных, типы данных, ограничения целостности и т. п. Очень часто пользователи на предприятиях такими знаниями не обладают.

Кроме того, существуют разные варианты использования Access с точки зрения архитектуры приложения. Иногда Access (файл MDB) используется просто как ядро, которое управляет данными, находящимися в таблицах. Пользователи работают с этими данными через внешние приложения, созданные разработчиками, например, на Visual Basic, Delphi или C++. В других ситуациях Access, наоборот, используется только для предоставления пользовательского интерфейса для работы с данными, которые физически расположены на серверах баз данных, например, SQL Server, Oracle, IBM D2 и т. п.

В Access предусмотрен встроенный язык запросов JET SQL, который активно используется разработчиками для работы с данными в базах Access.

Подводя итоги, можно сказать, что программирование средствами VBA в Access, которое будет рассмотрено в этой главе, — это лишь малая часть возможностей этого приложения. Очень многие возможности работы с Access (например, его язык запросов или проектирование и создание таблиц) останутся за пределами этой книги. Для этого существует отдельная литература. Заметим, что во многих книгах по Access за рамками остается как раз язык VBA и объектная модель самого Access, так что эта глава может послужить хорошим дополнением к ним.

Какие задачи на предприятии чаще всего решаются средствами автоматизации в Access?

Сразу же скажем, что поскольку Access — это система управления базами данных, то он очень часто используется как контейнер для хранения данных. При этом данные могут быть самыми разными, например, обычные данные о заключенных договорах или клиентах вашего предприятия, цифровые фотографии, шаблоны Word и Excel, которые используются для генерации отчетов из баз данных. В Access все эти данные вместе с графическим интерфейсом можно "упаковать" в один MDB-файл, что позволяет сделать приложение очень компактным и удобным для переноса с одного компьютера на другой.

Еще одно важное назначение Access — обеспечение клиентского интерфейса для работы с данными, которые хранятся на мощных клиент-серверных системах, таких как SQL Server, Oracle, IBM DB2. Согласно Microsoft, рекомендуется использовать настольные системы (такие как Access, FoxPro, Paradox и т. п.), если к данным одновременно будут обращаться не более 10 пользователей. Если пользователей больше или самих данных очень много (несколько гигабайт), то рекомендуется использовать более сложные, но и более функциональные клиент-серверные системы. А уже в рамках обеспечения доступа к данным (на клиент-серверных системах или прямо в базах данных Access) решаются более узкоспециализированные задачи приложений:

- *создание обычных форм*, т. е. формирование программных интерфейсов для занесения, изменения или просмотра данных в базе и Web-форм (они называются страницами доступа к данным);
- *создание отчетов к базам данных*, в том числе параметризованных;
- *создание программной логики приложения* обычным способом — на VBA (модули) и для начинающих пользователей (макросы, которые всегда можно преобразовать в модули);
- *вспомогательные действия* — печать, экспорт и преобразование данных (хотя для преобразования данных обычно удобнее использовать объектную модель DTS), загрузка данных, репликация и т. п.

12.2. Основные этапы создания приложений в Access

Если вы будете выступать в роли профессионального разработчика и создавать приложение, работающее с информацией на источнике данных (в базе данных Access или на клиент-серверной системе), то имеет смысл последовательно выполнить все рекомендованные далее этапы проектирования и создания таких приложений. Этим вы, возможно, убережете себя от ошибок, которые будет впоследствии трудно исправить.

- Первый этап — *сбор информации о потребностях предприятия*, *его подразделений и пользователей*: с какими унаследованными системами они работают, с чем нужно обеспечивать совместимость, как организованы информационные потоки, какова изменчивость системы и т. п. Часто при этом используются специальные программные средства, такие как Microsoft Visio и ERWin.
- Второй этап — *выбор архитектуры приложения*, *выбор подходящей системы управления базами данных и проектирование СУБД*. На этом этапе определяется, будет ли информация храниться с использованием ядра Jet, на котором работает Access, или клиент-серверной системы, проектируется система таблиц для хранения информации и отношений между ними, проектируются некоторые другие объекты базы данных. Кроме того, определяется архитектура приложения — сколько в ней будет уровней, будут ли использоваться терминальные или Web-технологии, будет ли применяться репликация и т. п.
- Третий этап — *реализация СУБД и бизнес-логики приложения*. На этом этапе проектируются, создаются, настраиваются и заполняются исходными данными объекты базы данных: таблицы, представления, хранимые процедуры, формы, отчеты, макросы, программные модули и т. п. При создании приложений в Access бóльшая часть этих операций выполняется при помощи графического интерфейса разработчика. Код VBA используется для проверки вводимых пользователем значений, для работы с элементами управления на форме, для переключения между формами, отчетами, другими элементами управления, для обращения к внешним объектным моделям и т. п. На этом этапе опять-таки могут помочь Visio и ERWin.
- Четвертый этап — *оптимизация производительности базы данных*, что часто упускается разработчиками. Задача эта комплексная, но включает в себя в том числе и оптимизацию кода VBA.
- Пятый этап — *тестирование и отладка приложения (см. гл. 6)*.
- Шестой этап — *развертывание приложения*.

Как мы видим, эти этапы достаточно комплексные, и рассмотреть их все в этой книге не представляется возможным. Поэтому в данной главе будет рассказано только про те моменты, в которых нам может помочь VBA.

12.3. Объект *Application*, его свойства и методы

Объектная модель Access по своей архитектуре сильно отличается от объектных моделей Word и Excel. Возможно, это объясняется тем, что Access — не "родной" для Microsoft, как Word и Excel, а приобретенный продукт третьей фирмы.

Один из немногих моментов, в котором программирование в Access похоже на программирование в Word и Excel — это наличие объекта `Application`, который находится на вершине иерархии объектной модели Access. Он точно так же может использоваться для программного запуска Access из других приложений, и его свойства и методы доступны из любой части кода. Запуск Access из другого приложения может выглядеть так:

```
Dim appAccess As Object
Set appAccess = CreateObject("Access.Application")
appAccess.Visible = True
MsgBox appAccess.Name
```

Если запустить такой код из процедуры Word или Excel, то по умолчанию по завершении работы этой процедуры объект будет удален из оперативной памяти (поэтому здесь и поставлено окно сообщения, чтобы этот процесс задержать).

Можно программно запускать Access еще множеством разных способов — через объектную модель Windows Explorer, через командный интерпретатор операционной системы, через текстовый ярлык mad, через API и т. п.

На практике программным образом запускать Access приходится достаточно редко, поскольку обычно удобнее оболочку приложения, запускающего Word, Excel и т. п., делать именно в Access. Открывать программным способом Access для доступа к данным базы в MDB-файле не рекомендуется, для этого лучше использовать объекты ADO, более простые и менее ресурсоемкие.

Теперь поговорим о свойствах и методах объекта `Application`. Как можно убедиться, их набор в Access мало похож на соответствующий набор в Word и Excel.

- `AutomationSecurity` — это свойство позволяет определить уровень безопасности при открытии базы данных. По умолчанию используется значе-

ние msoAutomationSecurityByUI — использовать то, что настроено на графическом экране через меню **Макрос | Безопасность**. Можно вообще запретить открытие файлов баз данных с макросами (msoAutomationSecurityForceDisable), но чаще используется значение msoAutomationSecurityLow, которое позволяет открыть файл данных без лишних вопросов.

- BrokenReference — позволяет проверить, есть ли разорванные ссылки (когда ваше приложение не может найти модуль dll или другую базу данных). Наличие конкретной ссылки можно проверить при помощи объекта Reference.
- CodeContextObject — очень полезное свойство, которое позволяет определить, из какого объекта базы данных (формы, отчета и т. п.) был запущен модуль или макрос. Это свойство можно, например, использовать для обнаружения источника ошибок.
- CodeData — еще одно важнейшее свойство. Оно позволяет получить доступ к коллекциям AllDatabaseDiagrams, AllFunctions, AllQueries, AllStoredProcedures, AllTables и AllViews. Правда, в этих коллекциях находятся одни и те же объекты AccessObject. На первый взгляд возможностей у них не так много, но на самом деле при помощи этого объекта мы можем настраивать десятки свойств для таблиц, запросов, диаграмм и других объектов базы данных Excel. Пример использования свойства CodeData для получения информации о всех таблицах в базе данных может выглядеть так:

```
For Each oTable In CodeData.AllTables
    Debug.Print oTable.Name
Next
```

- CodeProject — используется для тех же целей, что и CodeData, но предоставляет доступ к коллекциям программных модулей AllForms, AllReports, AllMacros, AllModules и AllDataAccessPages.
- CommandBars — возвращает коллекцию объектов CommandBar, т. е. панелей инструментов Access. Эту коллекцию можно использовать для настройки пользовательского интерфейса приложения, построенного на основе Access.
- CurrentData — действует аналогично CodeData и CodeProject. Это свойство позволяет получить доступ к тем же коллекциям, включая полученные с внешнего источника данных (SQL Server). Аналогично работает свойство CurrentProject.
- CurrentObjectName и CurrentObjectType — позволяют определить, какой объект на момент запуска процедуры находился в фокусе (из какого объ-

екта был вызван данный код). Эти свойства, конечно, удобно использовать для проверок, когда один и тот же код может быть вызван разными способами. При этом свойство `CurrentObjectType` ведет себя несколько неожиданно. В обычных ситуациях оно возвращает значение из перечисления, но если вызвавшего объекта уже нет (объектная ссылка установлена в `Nothing`) или информацию о нем получить не удалось, это свойство почему-то возвращает `True`.

- `DataAccessPages` — позволяет получить ссылку на одноименную коллекцию, в которой находятся объекты всех Web-форм базы данных (они называются страницами доступа к данным) — объектов `DataAccessPage`.
- `DBEngine` — позволяет получить ссылку на объект `DBEngine`, при помощи которого можно просмотреть или настроить свойства ядра Jet, на котором работает Access. Например, при помощи этого свойства можно сжать базу данных, настроить для нее пароль, определить используемую кодировку и т. п.
- `DoCmd` — позволяет получить доступ к еще одному очень важному объекту — `DoCmd`, при помощи которого можно выполнить множество важных операций. Фактически этот объект — основная "рабочая лошадка" Access с точки зрения VBA. Он будет рассмотрен в *разд. 12.4*. Сам объект `DoCmd` создавать нет необходимости — он всегда доступен через свойство объекта `Application`, например:

  ```
  Application.DoCmd.RunSQL "Delete from table1"
  ```

- `Forms` — позволяет вернуть ссылку на коллекцию объектов `Form`. От уже упомянутой коллекции `AllForms` эта коллекция отличается двумя особенностями:
  - в ней находятся только открытые в настоящий момент формы;
  - в ней находятся не объекты `AccessObject`, как в `AllForms`, а объекты `Form` с гораздо более богатым набором свойств и методов.
- `MenuBar` — позволяет не совсем стандартным образом настроить пользовательское меню (уровня всего приложения), вернуться к показу встроенного меню или вообще отключить показ меню. Для работы с контекстным меню предусмотрено свойство `ShortcutMenuBar`.
- `Modules` — действует аналогично свойству `Forms`. Оно позволяет получить доступ к одноименной коллекции с объектами `Module`, представляющими стандартные модули и модули классов. У объектов `Module` намного больше свойств и методов, чем у объекта `AccessObject`.
- `Printers` и `Printer` — возвращают соответственно коллекцию всех установленных в системе принтеров и принтер по умолчанию. При помощи

объекта Printer можно настроить множество свойств принтера, которые будут использоваться при печати.

- References — позволяет получить ссылку на коллекцию объектов Reference (ссылки на библиотеки типов). Можно использовать для проверки работоспособности ссылок или для добавления ссылок программным способом.
- Reports — работает аналогично свойствам Forms и Modules, позволяя получить доступ к коллекции объектов Report.
- Screen — это свойство возвращает специальный объект Screen, при помощи которого можно программным образом обратиться к активным элементам Access: формам, отчетам, элементам управления и т. п. (в Word и Excel свойства с префиксом Active... встроены непосредственно в объект Application). При помощи этого объекта можно также изменить вид курсора мыши и вернуться к предыдущему элементу управления.
- UserControl — позволяет определить, как именно был запущен Access: вручную пользователем или программным способом из другого приложения (в зависимости от этого, к примеру, можно решить, закрывать Access автоматически или положиться на пользователя).
- Version — позволяет вернуть версию Access (например, для дополнительных проверок работоспособности приложений). Для Access 2003 эта версия — 11.0.
- Visible — позволяет сделать Access видимым для пользователя или, наоборот, спрятать. По умолчанию Accesss, запущенный программным способом, невидим для пользователя, и это свойство необходимо установить в True.

Методов у объекта Application также очень много (кроме того, часть методов искусственно перенесена в объект DoCmd). Далее представлены самые важные из них.

- AccessError() — очень важный метод для обработки ошибок. Он позволяет получить описание ошибок библиотек Access и DAO, для которых стандартное Err.Description возвращает "Application-defined or object-defined error".
- BuildCriteria() — позволяет очень быстро и удобно сконструировать критерий отбора записей, который может применяться в SQL-запросах, фильтрах для формы и отчетов, и т. п. Возвращает правильно сконструированное строковое значение.
- CloseCurrentDatabase() — закрывает текущую базу данных без закрытия Access. Обычно применяется для того, чтобы затем открыть другую базу данных без запуска нового экземпляра Access.

- `CodeDb()` — возвращает объект `DAO.Database`, представляющий базу данных, в которой в настоящее время выполняется код (обычно используется, когда у вас есть специальная библиотечная база данных с программными модулями, выполняющими различные операции с другими базами данных). Ссылку на такой же объект для текущей базы данных можно получить при помощи метода `CurrentDb()`.
- `CompactRepair()` — позволяет сжать или починить базу данных Access и вернуть код ошибки (можно также записать протокол). Сжимаемая или ремонтируемая база данных должна быть в это время закрыта.
- `ConvertAccessProject()` — позволяет выполнить еще одну служебную операцию, на этот раз по преобразованию версии базы данных Access. Возможны варианты от `acFileFormatAccess2` до `acFileFormatAccess2002`.
- `CreateAccessProject()` — позволяет программным образом создать проект Access (так называется программный интерфейс для доступа к SQL Server). Для того чтобы его сразу создать и открыть, можно использовать метод `NewAccessProject()`, а чтобы просто открыть существующий — `OpenAccessProject()`.
- `CreateAdditionalData()` — позволяет создать объект `AdditionalData`, который можно использовать вместе с методом `ExportXML()` при экспорте родительской таблицы в XML-файл. Применение этого объекта позволяет экспортировать набор таблиц.
- `CreateControl()` — позволяет программным образом создать элемент управления на форме. Принимает множество параметров, которые определяют данный элемент управления. Для создания элемента управления в отчете используется метод `CreateReportControl()`. Удалить элемент управления можно при помощи соответственно `DeleteControl()` и `DeleteReportControl()`.
- `CreateForm()` — позволяет также программным образом создать форму Access и получить ссылку на объект созданной формы. Затем можно настроить свойства этой формы, добавить для нее элементы управления и т. п. Создание таким же образом отчета можно произвести при помощи метода `CreateReport()`.
- `CreateGroupLevel()` — позволяет программным образом создать группировку в отчете или отсортировать записи. Принимает имя отчета, столбец, по которому производится сортировка, формулы для создания верхнего и нижнего колонтитула групп.
- `CreateNewWorkgroupFile()` — позволяет программным способом создать файл рабочей группы с разрешениями для пользователей. На графическом

экране эту операцию можно выполнить при помощи пункта меню **Сервис | Защита | Администратор рабочих групп**.

- `CurrentUser()` — этот метод позволяет получить в виде строкового значения имя текущего пользователя базы данных. По умолчанию работа производится от имени пользователя Admin.
- Методы с префиксом `D...` — очень удобны для выполнения различных операций, не прибегая к коду SQL, напрямую из Access:
  - `DAvg()`, `DSum()`, `DCount()`, `DMax()`, `DMin()` и т. п. — позволяют применить агрегатные функции к столбцу (или набору записей) в таблице или представлении;
  - `DLookup()` — исключительно удобный метод, который позволяет найти и вернуть нужное вам значение из таблицы или представления (включая двоичные объекты, например шаблоны Word). Принимает в качестве параметров имя таблицы или представления, имя столбца и фильтр. Если условию фильтра удовлетворяет несколько значений, то возвращается первое из них;
  - `DFirst()` и `DLast()` — несмотря на свои названия, эти методы работают одинаково, возвращая случайное значение из столбца таблицы или представления.
- `Echo()` — позволяет перерисовать экран Access, а также вывести текст в строку состояния.
- `Eval()` — этот метод очень удобен во многих ситуациях. Он позволяет произвести над текстовой строкой операции, как будто это строка кода VBA. Этот метод возвращает значение типа `Variant`, чтобы подходили любые возвращаемые значения. Например:

  ```
  Eval("1 + 1")
  ```

  вернет 2, а вызов:

  ```
  Eval("Моя_Функция()")
  ```

  вернет то, что возвращает эта функция. `Eval()` очень удобно использовать, чтобы избежать громоздких проверок и преобразований типов, например, когда мы принимаем разные значения, вводимые пользователем.
- `ExportXML()` и `ImportXML()` — позволяют экспортировать и импортировать наборы таблиц с данными (включая информацию о ключах, индексах, кодировках и т. п.) в XML-совместимый текстовый файл. При этом экспорт и импорт из Access при помощи этих методов можно производить не только для баз данных Access, но и баз данных SQL Server, начиная с версии 6.5.

- `GetOptions()` и `SetOptions()` — позволяют получить информацию и установить те настройки, которые доступны через меню **Сервис | Параметры**. Например, чтобы при нажатии клавиши <Enter> в таблице производился переход не вправо (по умолчанию), а вниз, можно использовать код:

```
Application.SetOption "Move After Enter", 2
```

- `hWndAccessApp()` — очень нужный метод для тех, кто работает с Windows API. Позволяет вернуть указатель на окно Access.
- `NewCurrentDatabase()` — позволяет создать и сразу открыть новую базу данных Access. Для открытия существующей базы данных можно использовать метод `OpenCurrentDatabase()`.
- `Nz()` — исключительно удобный метод для практической работы. Позволяет возвращать пустую строку или другое значение, если значение в данном столбце таблицы не определено (`Null`). Опытные разработчики очень часто используют эту функцию, чтобы избежать ошибок при обращении к пустым значениям (любым, включая `Memo`), например, при поиске по таблице.
- `Quit()` — закрывает Access. Может ничего не сохранять, сохранять все или спрашивать у пользователя.
- `RefreshDatabaseWindow()` — позволяет обновить окно базы данных. Обычно применяется после программного создания форм, отчетов и т. п.
- `Run()` — позволяет вызвать процедуру или функцию VBA из кода и передать ей до 30 параметров. Может использоваться для вызова пользовательских или встроенных функций, но поскольку их можно вызвать и стандартными способами, то чаще всего этот метод используется при вызове процедуры Access из внешней откомпилированной программы, например, DLL или EXE.
- `RunCommand()` — позволяет выполнить одну из десятков встроенных команд Access (практически все, что есть на панелях управления и во встроенных меню). Например, чтобы максимизировать окно Access, можно воспользоваться командой:

```
RunCommand acCmdAppMaximize
```

- `SysCmd()` — позволяет выполнить множество служебных операций: получить информацию о домашнем каталоге, о версии Access, о состоянии указанного вами объекта базы данных, запустить графический индикатор выполнения в строке состояния и т. п.

12.4. Макрокоманды и объект *DoCmd*

Объект `DoCmd` — это "рабочая лошадка" программирования в Access. Этот объект позволяет программным образом выполнять макрокоманды Access — те действия *(actions)*, которые можно просмотреть (на русском языке) в окне конструктора макрокоманд. Действия — это самые распространенные операции, которые обычно приходится выполнять в Access программным способом.

У объекта `DoCmd` нет свойств, только методы. Для целей унификации в последних версиях Access методы `DoCmd` "переезжают" в объект `Application`, но для совместимости со старыми приложениями они оставлены и в `DoCmd`. Microsoft рекомендует по возможности пользоваться одноименными методами объекта `Application`.

Приводить здесь методы `DoCmd` с комментариями нет никакого смысла — эти методы в точности соответствуют набору действий в конструкторе макрокоманд (макрокоманд чуть больше за счет того, что для некоторых из них — окно сообщений, запуск внешнего приложения, передача нажатий клавиш и т. п. — предусмотрены отдельные средства VBA).

Подробно описывать макрокоманды мы не будем: для каждой из них предусмотрено описание на русском языке и подробная справка на английском по нажатию клавиши <F1>. Просмотреть такое описание можно из окна создания макросов (для этого в окне базы данных нужно перейти на вкладку **Макросы** и нажать кнопку **Создать**. Писать код вручную для них также нет никакого смысла — всегда есть возможность преобразовать созданный макрос в модуль и просмотреть полученный код. Далее перечислены лишь основные возможности макрокоманд (методов объекта `DoCmd`).

- `OutputTo()` (соответствует макрокоманде `ВывестиВФормате`), `TransferText()` (`ПреобразоватьТекст`), `TransferDatabase()` (`ПреобразоватьБазуДанных`), `TransferSpreadsheet()` (`ПреобразоватьЭлектроннуюТаблицу`) — макрокоманды, которые обеспечивают экспорт и импорт данных (в формат Excel, RTF, SNP, TXT, DBF, с источниками данных ODBC и т. п.). У каждого из форматов есть свои особенности и недостатки. Можно использовать и рассмотренные нами ранее средства Word и Excel.
- `RunSQL()` (`ЗапускЗапросаSQL`), `RunMacro()` (`ЗапускМакроса`) — позволяют выполнить запрос на языке SQL или макрос соответственно.
- Методы с префиксом `Open...()` (макрокоманды с префиксом `Открыть...` — `Таблицу`, `Запрос`, `Представление`, `Форму` и т. п.) — их действия также понятны из названий. Можно выбрать режим открытия (конструктор, просмотр и т. п.) и многие другие параметры. После открытия объекта можно воспользоваться его кодом и его элементами управления.

Есть возможность также копировать базу данных и отдельные файлы, искать записи, активизировать элементы управления и выполнять множество других операций.

12.5. Работа с формами Access из VBA (объект *Form*)

Один из важнейших элементов Access, который широко используется в приложениях, — это формы. Формы Access предназначены для того же, для чего и обычные формы VBA — это прежде всего контейнеры для графических элементов управления. Но устройство форм Access, их функциональные возможности, приемы работы с ними и даже наборы элементов управления, которые на них можно размещать, сильно отличаются от привычных нам форм VBA, которые можно использовать в Word и Excel.

Формы Access используются:

1. *Для редактирования записей в таблицах базы данных Access и внешних источников данных*. Для того чтобы создать такие формы, вообще не нужно никакого программирования — достаточно создать форму в режиме конструктора или воспользоваться мастером создания форм. Подключиться к внешнему источнику данных (например, к базе данных SQL Server или Oracle) можно, воспользовавшись в Access меню **Файл | Внешние данные | Связь с таблицами**.
2. *Как панели управления вашего приложения.* Очень часто в приложении на основе Access создается начальная форма, которая открывается при запуске этого приложения. На этой формы предусмотрены кнопки и другие элементы управления для вызова других форм, отчетов, макросов, выхода из приложения и выполнения прочих операций. После закрытия других форм управление опять передается начальной форме.
3. *Просто для предоставления пользователю возможностей для выполнения любых действий.* Например, форму можно использовать для выбора пользователем параметров отчета, выгрузки данных во внешнее приложение (например, Excel) и т. п.

Как работать с формами Access из VBA?

Первое, что необходимо сказать — для работы с формами во многих ситуациях нам придется использовать общий объект `AccessObject`, который представляет в Access не только формы, но и таблицы, макросы, модули, отчеты и множество других элементов. Поскольку этот объект универсальный, то, конечно, большой помощи от подсказки в редакторе VBA у нас не будет. Обратиться к объекту формы можно через коллекцию `AllForms`, которая доступна

через объекты CodeProject и CurrentProject. Например, получить информацию о всех формах в базе данных Access можно так:

```
Dim oA As AccessObject
For Each oA In CurrentProject.AllForms
    Debug.Print oA.Name
Next
```

Если вы будете обращаться к формам в коллекции AllForms по индексу, обратите внимание, что нумерация форм в этой коллекции начинается с 0. Обращаться к элементам в этой коллекции можно и по имени:

```
Debug.Print CurrentProject.AllForms("Форма1").IsLoaded
```

Специальное свойство IsLoaded определяет, открыта ли эта форма (т. е. загружена ли она в оперативную память).

Программно формы можно найти и другим способом. Все открытые формы Access автоматически помещаются в коллекцию Application.Forms и представляются в виде объекта Form. Это уже нормальный объект, свойства которого соответствуют свойствам формы, доступным через графический интерфейс. Например, если форма Форма1 открыта, получить информацию о ее ширине можно так:

```
Debug.Print Application.Forms("Форма1").Width
```

Это свойство можно использовать и для изменения ширины формы, но для этой цели рекомендуется использовать метод DoCmd.MoveSize(), который изменяет размеры активного объекта (например, нашей формы, если она активна):

```
DoCmd.MoveSize Width:=10000
```

Еще одна возможность: если вы работаете с кодом самой формы или ее элементов управления (например, внутри события Click кнопки, которая расположена на форме), то обратиться к объекту самой этой формы можно просто, используя ключевое слово Form.

Как открыть форму?

Если в Word или Excel нам обязательно нужно открывать форму программным способом, то в Access это делать совсем необязательно. Можно открыть форму и вручную из окна базы данных (рис. 12.1). Из этого же окна обычно производится создание новых форм или изменение уже существующих.

Еще один часто используемый способ — это просто запустить форму при открытии базы данных Access. Для этого в меню **Сервис** нужно выбрать пункт **Параметры запуска** и выбрать нужную форму в списке **Вывод формы/страницы**. Если при этом вы уберете все остальные флажки, то приложение при открытии может выглядеть так, как показано на рис. 12.2.

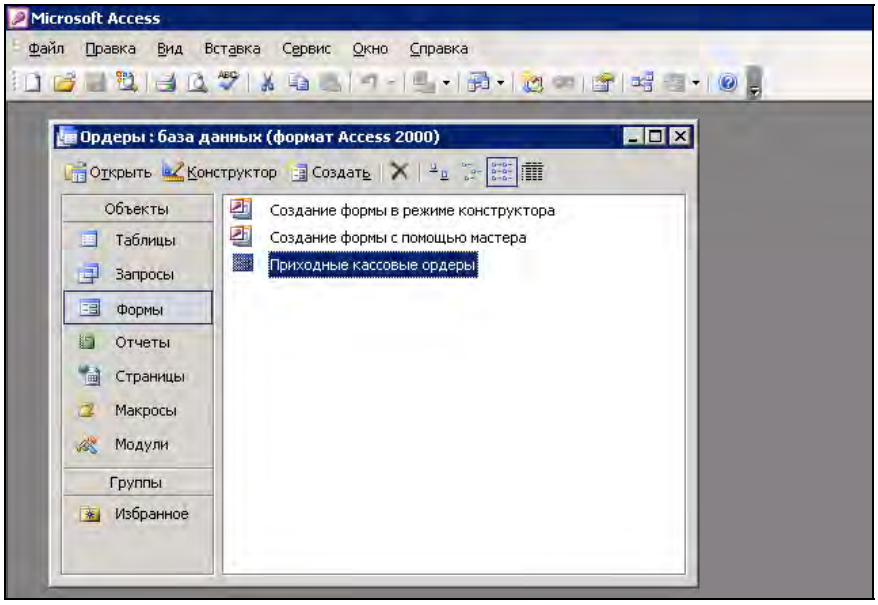


Рис. 12.1. Окно базы данных

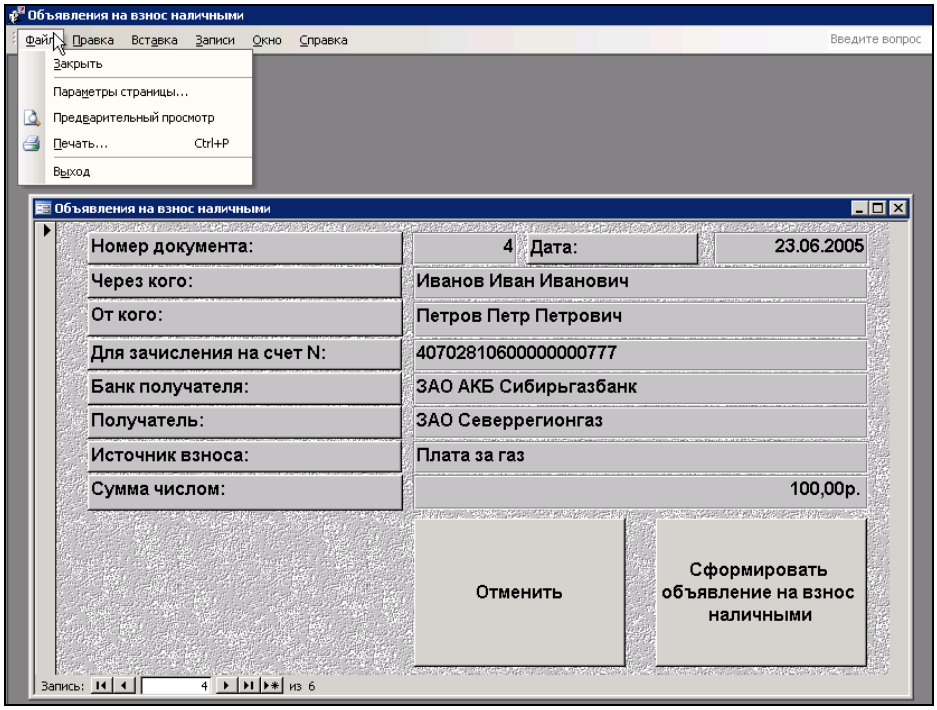


Рис. 12.2. При открытии Access пользователь видит только одну форму

Окно базы данных будет скрыто от пользователя, а это значит, что пользователь не сможет получить информацию ни о таблицах нашей базы данных, ни о других формах, ни о модулях — все служебные элементы базы данных будут от него спрятаны. В принципе, пользователь может обойти такую защиту, держа нажатой при запуске Access клавишу <Shift>, но программным способом можно закрыть и такую возможность.

Если все-таки нужно открыть форму программно (например, из другой формы), то для этой цели можно использовать метод `DoCmd.OpenForm()`. В самом простом варианте этот метод просто принимает параметр с именем формы:

```
DoCmd.OpenForm "Форма1"
```

Если форма уже открыта, то этот метод не открывает ее заново, а просто активизирует. Метод `DoCmd.OpenForm()` принимает также несколько необязательных параметров, при помощи которых вы можете настроить фильтр на отображение записей в форме, режим открытия формы (например, модальность) и т. п. Закрытие формы производится при помощи метода `DoCmd.Close()`. Если же вам нужно просто спрятать форму, чтобы сохранить введенные на ней пользователем значения и отобразить их при следующем показе, можно просто сделать форму невидимой, назначив ее свойству `Visible` значение `False`.

Форма нам обычно нужна как контейнер для расположенных на ней элементов управления. Обычно элементы управления программным способом создавать не нужно, намного проще и удобнее поместить их на форму в режиме конструктора. В наборе элементов управления для формы предусмотрены как знакомые нам элементы управления (текстовые поля, надписи, кнопки, флажки и переключатели), так и новые (свободная и присоединенная рамки объектов, разрывы страниц, подчиненные формы/отчеты и т. п.). В верхнем правом углу **Панели инструментов** в конструкторе формы Microsoft Access находится специальная кнопка **Мастера**. Если она нажата, то добавление на форму привычных элементов управления (например, кнопки) приведет к появлению окна мастера, который попытается помочь вам автоматически сгенерировать нужный код VBA для этого элемента управления (рис. 12.3).

Можно использовать генерируемый мастером код как заменитель макрорекордера (которого в Access нет), чтобы понять, как можно выполнить те или иные действия.

В Access добавлены нестандартные (по отношению к обычным формам VBA) элементы управления.

- **Свободная рамка объекта** — предоставляет возможность разместить на форме OLE-объект (например, документ Word, лист Excel, презентацию PowerPoint, рисунок, звукозапись или видеоклип), который может быть

встроен в базу данных Access (но не помещен в таблицу) или находиться во внешнем по отношению к базе данных Access файлу.

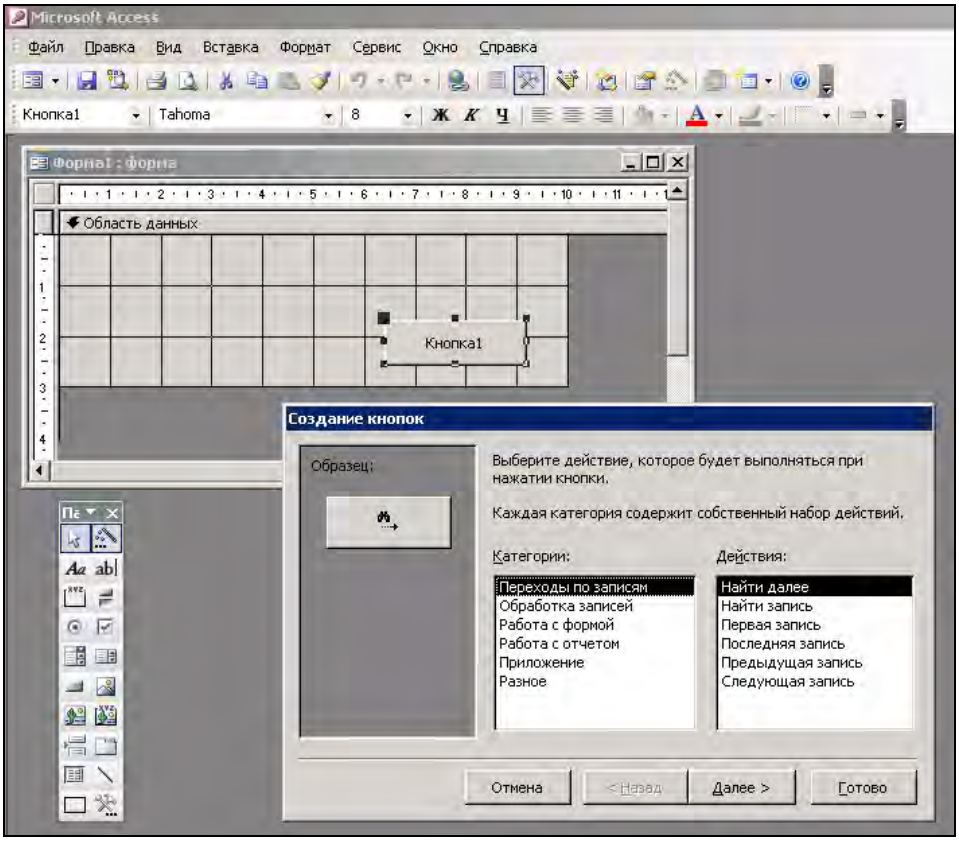


Рис. 12.3. Мастер создания кнопок

- **Присоединенная рамка объекта** — то же самое, за исключением того, что этот элемент применяется для работы с объектами OLE, которые хранятся в таблицах внутри баз данных Access или внешнего источника данных. Это самый удобный способ генерации отчетов в Word.

  Предположим, что в нашей базе данных Access находится таблица с тремя столбцами, как показано на рис. 12.4.

  В столбце **File** у нас хранятся шаблоны Word, которые используются для генерации отчетов. Мы помещаем на форму элемент управления **Присоединенная рамка объекта** с именем `WordTemplate`. После этого все, что нужно для создания файла Word на основе шаблона из базы данных, у нас уже есть.

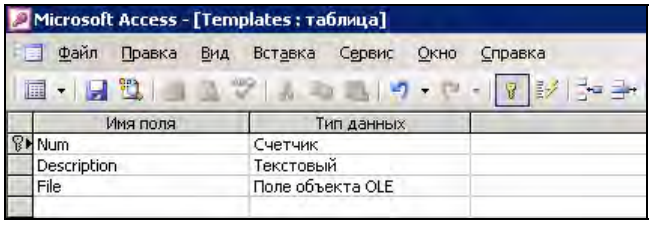


Рис. 12.4. Таблица для хранения шаблонов Word

Для кнопки, по нажатию на которой будет формироваться отчет, можно использовать следующий код:

```
'Получаем ссылку oFrame на объект присоединенной рамки на форме
Dim oFrame As BoundObjectFrame
Set oFrame = oForm.Controls("WordTemplate")

'При помощи метода DLookup() скачиваем в него значение столбца File
'из таблицы Templates, где номер строки (значение столбца Num) = 1
oFrame = Application.DLookup("[File]", "Templates", "[Num] = 1")

'Открываем объект в отдельном окне приложения, т. е. создаем
'документ Word на основе шаблона, загруженного в рамку объекта на форме
oFrame.Verb = acOLEVerbOpen

'Активизируем объект приложения
oFrame.Action = acOLEActivate

'Получаем ссылку на Word в переменную oWord
Dim oWord As Word.Application
Set oWord = GetObject(, "Word.Application")

'Получаем ссылку на созданный нами документ
Dim oDoc As Word.Document
Set oDoc = oWord.ActiveDocument

'Дальше работаем средствами Word, например, вставляем нужный текст
'в места, отмеченные закладками
```

Конечно же, правильнее будет при этом сделать эту присоединенную рамку объекта на форме изначально невидимой, чтобы пользователь не мог активизировать этот объект по собственной инициативе.

- **Разрыв страницы** — этот элемент управления определяет начало нового экрана формы.
- **Подчиненная форма/отчет** — используется для размещения на форме подчиненных форм, таблиц или отчетов.

Как уже говорилось, программным способом элементы управления в форме Access приходится создавать редко. Если на форме вам нужен переменный набор элементов управления, то правильнее будет с самого начала добавить на форму все нужные элементы управления и по необходимости делать их то видимыми, то невидимыми. Тем не менее создать программным способом элементы управления на форме тоже можно. Эта операция выполняется при помощи метода `Application.CreateControl()`, который принимает множество параметров: имя формы, на которой создается элемент управления, тип элемента управления, его месторасположение на форме и т. п.

Обращение к значениям элементов управления на форме производится через коллекцию `Controls`, которая умеет работать с именами элементов управления:

```
cSumInNumber = oForm.Controls("SumNumber").Value
```

12.6. Свойства, методы и события форм

Далее для справки приводится информация о самых важных свойствах, методах и событиях, которые разработчики предусмотрели для объекта `Form` в Access.

Этот объект имеет следующие свойства.

- `ActiveControl` — позволяет определить, в каком элементе управления формы в данный момент находится фокус. Обычно используется для проверок. Это свойство, помимо формы, есть также у отчета и специального объекта `Screen`. При помощи свойства `Screen.ActiveControl` можно определить не только имя активного в настоящий момент элемента управления, но и к какой форме или отчету он относится, например:

  ```
  MsgBox Screen.ActiveControl.Parent.Name
  ```

- Свойства с префиксом `After...` — позволяют заменить обычные событийные процедуры, назначив имя процедуры какому-либо событию (`AfterInstall` — вставка новой записи, `AfterUpdate` — изменение существующей записи и т. п.).

- `AllowAdditions` — определяет, разрешается или нет пользователю добавлять новые записи через данную форму. Для того чтобы разрешить или запретить удаление записей, используется свойство `AllowDeletions`, для разрешения или запрета редактирования существующих записей — свойство `AllowEdits`. Остальные свойства с префиксом `Allow...` относятся к разрешению или запрету для пользователя использовать различные режимы отображения информации.

- `AutoCenter` — определяет, будет ли форма выводиться точно по центру окна приложения.
- `AutoResize` — определяет, будет ли форма автоматически изменять свои размеры, подстраиваясь под размеры записей на ней. Это свойство нужно использовать очень осторожно, поскольку при таком автоматическом изменении размеров пользователь может увидеть на экране совсем не то, что вы задумывали.
- Свойства с префиксом `Before...` — эти многочисленные свойства, так же как и `After...`, заменяют событийные процедуры форм.
- `Bookmark` — очень полезное свойство. Работает точно так же, как и одноименное свойство объекта `Recordset` в ADO *(см. гл. 9)*. Оно позволяет сохранить информацию о текущей записи (закладку на нее) в строковой переменной. Если мы присвоим этому свойству значение, сохраненное в строковой переменной, то форма вернется на ту запись, для которой была сохранена закладка.
- `BorderStyle` — позволяет определить рамку на форме (напрямую, без вложенных объектов).
- `Caption` — определяет заголовок формы.
- `ChartSpace` — представляет специальный контейнерный объект, который может содержать в себе до 64 диаграмм. Используется для программного создания диаграмм на формах и для программного изменения существующих диаграмм.
- `CloseButton` — позволяет не возиться со свойствами кнопки **Close**, а делать ее то доступной, то недоступной для пользователя прямо из свойств формы. Используется, конечно, чтобы не дать пользователю прервать выполнение каких-то действий (например, ввода данных) на полпути.
- `ControlBox` — позволяет добавить или убрать меню **Control** (значок в левой части заголовка формы с командами **Переместить**, **Свернуть**, **Развернуть** и т. п.). Если убрать это меню, то исчезнут и специальные иконки в правом углу заголовка формы — **Свернуть**, **Развернуть** и **Закрыть**. Обычно это свойство используется для специальных модальных форм, например, выполняющих роль окон предупреждений. К сожалению, настраивать это свойство можно только тогда, когда форма открыта в режиме конструктора. Схожим образом работает свойство `MinMaxButtons`, которое позволяет определить, будут ли на форме видны кнопки **Свернуть** и **Развернуть** (тоже только в режиме конструктора).
- `Controls` — одно из важнейших свойств. Возвращает коллекцию `Controls` со всеми элементами управления на данной форме.

- `Count` — возвращает количество элементов управления на форме. Можно сказать, что относится к коллекции `Controls`, а не к самой форме.
- `CurrentRecord` — возвращает номер текущей записи из числа всех записей, которые открыты на форме. Это свойство доступно только для чтения.
- `CurrentView` — очень важное и полезное свойство. Оно позволяет определить, в каком режиме открыта данная форма: в режиме конструктора (**Design View**), в режиме формы (**Form View**) или в виде таблицы данных (**Datasheet View**). Свойство доступно только для чтения. Для переключения между режимами нужно использовать методы `DoCmd.Close()` и `DoCmd.OpenForm()` с соответствующими параметрами.
- `Cycle` — это свойство позволяет определить, что будет, когда пользователь нажмет на клавишу <Tab>, находясь на последней записи по порядку перехода: перейдет ли он на первый элемент управления для следующей записи (по умолчанию), вернется ли на первый элемент управления для текущей записи или будет ходить по кругу в рамках текущей страницы (экрана) формы.
- `DataEntry` — позволяет перевести форму в хитрый режим ввода новых данных, когда для пользователя на форме будет доступна только одна новая запись — пустая. Никакие уже существующие записи пользователю видны не будут.
- Свойства с префиксом `DataSheet...` — эти многочисленные свойства позволяют определить внешний вид формы, когда она открыта в режиме **Datasheet View**, т. е. в виде таблицы.
- `DefaultControl` — предоставляет интересную возможность, которая может сильно сократить количество набираемого кода, если вы создаете форму программным способом. При этом вы можете определить параметры для виртуального элемента управления `DefaultControl` любого типа. После того, как вы создадите новый элемент управления такого же типа, к нему будут автоматически применены те настройки, которые вы определили для `DefaultControl`. Такая возможность может быть очень удобной, например, если на форме вам программным способом придется создать десяток текстовых полей с одинаковым шрифтом, цветом фона и т. п. Например:

```
'Создаем текстовое поле по умолчанию
Set oDefaultTextBox = oForm.DefaultControl(acTextBox)
oDefaultTextBox.FontSize = 12
'Создаем новое текстовое поле на форме — уже с размером шрифта 12
Set oTextBox1 = CreateControl(oForm.Name, acTextBox, , , , 500, 500)
```

- `DefaultView` — определяет, в каком режиме форма будет открываться по умолчанию. Пользователь сможет переключиться в другой режим, если это будет разрешено ему при помощи свойства `ViewsAllowed`.
- `Dirty` — принимает значение `True`, если текущая запись была изменена, но еще не сохранена в базе данных. Как только текущая запись будет сохранена, значение этого свойства опять изменится на `False`. Обычно используется для проверок во избежание потери введенных пользователем данных.
- `DividingLines` — определяет, будут ли на форме разделительные линии, отделяющие друг от друга области формы или записи, если их на форме очень много.
- `FetchDefaults` — определяет, будут ли на форме выводиться значения по умолчанию для столбцов при заполнении новой записи.
- `Filter` — исключительно важное свойство. Оно позволяет отфильтровать записи на форме (например, по диапазону дат, выбранных пользователем, по значению какого-либо столбца, которое пользователь может, например, выбрать при помощи раскрывающегося списка и т. п.). В качестве значения этому свойству передается то выражение, которое должно идти после ключевого слова `Where` в запросах на языке SQL. Вместе с этим свойством обычно для включения фильтра используется свойство `FilterOn`. Например, если форма привязана к таблице со столбцом `City` и нужно в форме отобразить только те записи, для которых в этом столбце значится "`Санкт-Петербург`", можно использовать следующий код:

  ```
  oForm.Filter = "City = 'Санкт-Петербург'"
  oForm.FilterOn = True
  ```

  Учтите, что это свойство используется для фильтрации записей только в самой форме. Если вы обращаетесь при помощи формы к данным в базе данных SQL Server или Oracle, то чаще всего бывает выгоднее применить фильтр в запросе, передаваемом на сервер (чтобы не скачивать в приложение лишние данные). Для этой цели используется свойство `ServerFilter` *(см. далее)*.
- `Form` — это свойство позволяет получить ссылку на форму или отчет, которые помещены на форму при помощи элемента управления **Подчиненная форма/отчет**.
- `FrozenColumns` — позволяет определить количество закрепленных столбцов, т. е. тех столбцов, которые не будут уходить за экран при прокрутке формы вправо. Это свойство доступно только для чтения. Для настройки параметров в форме предлагается использовать на графическом экране команду **Закрепить столбцы** в меню **Формат**.

- `HasModule` — это свойство определяет, есть ли у данной формы или отчета свой собственный модуль класса (по умолчанию нет). При помощи пользовательского модуля класса можно добавить для формы свои собственные свойства и методы или переопределить существующие. Обычно такая возможность, как и работа с пользовательскими классами в VBA, используется только в очень больших и сложных проектах со специфическими требованиями.
- `HelpFile` и `HelpContextId` — позволяют определить файл справки и закладку в нем для данной формы.
- `Hwnd` — позволяет вернуть дескриптор окна Windows для данной формы. Используется при работе с Windows API.
- `InputParameters` — позволяет вернуть (в виде строкового значения) информацию о параметрах, которые были переданы на источник данных при выполнении запроса, результаты которого были загружены в форму.
- `KeyPreview` — это свойство определяет, кому будут передаваться нажатия клавиш: только активному элементу управления на форме (по умолчанию) или вначале форме, а затем уже активному элементу управления. При помощи этого свойства можно обрабатывать событийными процедурами для формы клавиатурные комбинации, в каком бы месте на форме не находился фокус, например, так:

```
Private Sub Form_Load()
    Me.KeyPreview = True
End Sub

Private Sub Form_KeyDown(KeyCode As Integer, Shift As Integer)
    Select Case KeyCode
        Case vbKeyF5
            MsgBox "Нажата клавиша F5!"
        Case vbKeyF6
            MsgBox "Нажата клавиша F6!"
    End Select
End Sub
```

- `MenuBar` — позволяет указать имя пользовательского меню, которое будет автоматически появляться при открытии данной формы или отчета (см. также свойства `ShortcutMenuBar` и `Toolbar`).
- `Modal` — очень важное свойство. Если для него будет установлено значение `True`, то форма станет модальной. Это значит, что ее придется закрыть, прежде чем фокус можно будет передать другой форме. Чтобы отключить еще и панели инструментов и меню на время работы формы, нужно установить также значение свойства `Popup` в `True`.

- `Module` — это свойство позволяет получить ссылку на объект модуля класса для данной формы (см. далее свойство `HasModule`). С этим свойством нужно быть осторожным: если форма находится в режиме конструктора, то обращение к нему автоматически приведет к созданию нового модуля класса для формы (если для этой формы модуля класса еще не существовало).
- `Moveable` — позволяет определить, сможет ли пользователь перемещать форму на экране. На программное изменение местонахождения формы не влияет: программным способом перемещать форму можно всегда.
- `Name` — это, конечно, имя формы, под которым она будет видна в окне базы данных.
- `NavigationButtons` — это свойство позволяет определить, будут ли в распоряжение пользователя предоставлены кнопки для перехода по записям (на первую, последнюю, новую, следующую и т. п.) в нижней части формы.
- `NewRecord` — позволяет определить, является ли текущая запись для формы новой записью. Это свойство используется для проверок.
- `ObjectPalette` и `PaintPalette` — позволяют определить для вновь создаваемых элементов на форме и для самой формы соответственно цветовую схему Windows. В отличие от системы Windows, Access позволяет использовать неограниченное количество цветовых схем одновременно. Цветовая схема будет сохранена вместе со схемой или отчетом. Свойство `PaletteSource` позволяет указать цветовую схему для формы или отчета в виде файлов на диске.
- Свойства с префиксом `On...` (`OnClick`, `OnClose`, `OnConnect` и т. п.) — эти многочисленные свойства, как `After...` и `Before...`, заменяют событийные процедуры для формы, позволяя назначить событию имя макроса Access или процедуры VBA.
- `OnTimer` — это свойство позволяет определить реакцию на события, которые автоматически генерируются Access через определенный интервал времени, определяемый при помощи свойства `TimeInterval`.
- `OpenArgs` — позволяет получить доступ к параметру, передаваемому форме при ее открытии методом `DoCmd.OpenForm()`.
- `OrderBy` — свойство, которое позволяет настроить сортировку записей в форме. Принимает строковое выражение, состоящее из имени столбца, по которому производится сортировка, и направления сортировки, например:

```
oForm.OrderBy = "City DESC"
oForm.OrderByOn = True
```

Непосредственно включение сортировки производится при помощи свойства `OrderByOn`.

- `Page` — возвращает информацию о номере текущей страницы формы или отчета при печати. Обычно используется в текстовых полях для генерации номеров страниц. Доступно только в режиме предпросмотра перед печатью или при печати. Совместно с ним часто используется свойство `Pages`, которое возвращает общее количество страниц в форме или отчете.
- `Painting` — позволяет запретить перерисовку данной формы или отчета. Обычно устанавливается в `False` на время выполнения длинной ресурсоемкой операции для сбережения ресурсов. После завершения такой операции это свойство нужно опять установить в `True`. Отключить перерисовку для всех окон Access можно при помощи макрокоманды `ВыводНаЭкран` (метода `Echo()` объекта `DoCmd`).
- Свойства с префиксом `Picture...` — позволяют поместить на форму изображение (обычно в виде фонового режима) и настроить его параметры.
- `PivotTable` — позволяет получить доступ к объекту `PivotTable`, если он размещен на форме. Объект `PivotTable` в Access — это Web-версия (компонент ActiveX) сводной таблицы, которая была рассмотрена в *разд. 11.8*.
- `PopUp` — это свойство, установленное в `True`, определяет, что форма будет открыта как всплывающее *(popup)* окно поверх всех других окон Access. При этом стандартные меню и панели инструментов Access станут недоступны. Обычно используется вместе со свойством `Modal`.
- `Printer` — позволяет получить информацию или (чаще всего) настроить принтер по умолчанию — для формы, отчета или всего приложения (это свойство есть и у объекта `Application`). Проверить, будет ли использоваться принтер по умолчанию, можно при помощи свойства `UseDefaultPrinter`.
- `Properties` — возвращает коллекцию объектов `Property`, представляющих все встроенные свойства формы или отчета. У этих объектов всего два своих свойства: `Name` (имя свойства) и `Value` (его значение). Обычно используется для того, чтобы пройти циклом по всем свойствам формы или отчета.
- Свойства с префиксом `Prt...` — предназначены для настройки принтера перед печатью формы или отчета. Принимают в качестве параметров достаточно сложные байтовые структуры. Рекомендуется работать с этими свойствами, только обладая полной информацией о возможностях драйвера принтера.
- `RecordLocks` — позволяет определить, как будет происходить блокировка записей в форме при одновременном редактировании данных через эту форму сразу несколькими пользователями: 0 — записи блокироваться не

будут (вы рискуете тем, что пользователь при попытке сохранения изменений в своей записи столкнется с сообщением об ошибке); 1 — все записи в таблице, на которую ссылается форма, будут заблокированы, остальные пользователи смогут их лишь просматривать; 2 — будет заблокирована страница с изменяемой записью (примерно 4 Кбайт в базе данных), остальным пользователям данные этой страницы будут доступны только для чтения.

- `RecordSelectors` — определяет, будет ли доступна в режиме формы (**Form View**) область выделения записи — прямоугольник в левой части формы, при помощи которого можно выделить всю запись.
- `Recordset` — позволяет вернуть или настроить объект `Recordset` из библиотек ADO *(см. гл. 9)* или DAO (унаследованная библиотека, которая в этой книге не рассматривается), который используется в качестве источника информации для формы. Применение этого свойства может быть очень удобным при обращении к внешним источникам данных. Кроме того, в вашем распоряжении появляются очень удобные и хорошо известные многим разработчикам свойства и методы объекта `Recordset`.
- `RecordsetType` — позволяет определить тип объекта `Recordset`, на котором основана форма, с точки зрения возможности редактирования его информации.
- `RecordSource` — позволяет определить источник данных для формы или отчета. Принимает текстовое значение, которое может быть именем таблицы или представления или запросом на языке SQL. Если вы изменили свойство `Recordset`, то значение этого свойства меняется автоматически.
- `RecordSourceQualifier` — используется только тогда, когда источник данных для формы Access — это база данных SQL Server. Возвращает имя владельца таблицы или представления, на которое ссылается форма.
- `ResyncCommand` — позволяет самостоятельно определить команду, которая пойдет на источник данных для записи изменений, внесенных пользователем через форму. Это свойство используется только тогда, когда команда на изменение данных, генерируемая автоматически, вас по какой-то причине не устраивает.
- `ScrollBars` — определяет, отображать ли на форме горизонтальную и вертикальную полосы прокрутки. По умолчанию обе полосы прокрутки на форме отображаются.
- `Section` — это свойство очень похоже на метод. Оно принимает параметр, обозначающий область (секцию) формы или отчета, и используется для настроек свойств этой области, например:

```
oForm.Section(acPageHeader).Visible = False
```

- `ServerFilter` — это свойство позволяет вставить фильтр в запрос, который при открытии формы отправляется источнику данных (например, базе данных SQL Server или Oracle). При этом синтаксическую правильность строкового выражения, которое вы передаете в качестве фильтра, Access не проверяет. Чтобы включить работу с серверным фильтром, необходимо установить значение свойства `ServerFilterByForm` в `True`:

  ```
  oForm.ServerFilter = "City = 'Санкт-Петербург'"
  oForm.ServerFilterByForm = True
  ```

  Удобнее всего помещать этот код в событийную процедуру для события `OnOpen` формы.

- `ShortcutMenu` — если установить значение этого свойства в `False`, то будет запрещен показ контекстных меню как для самой формы, так и для расположенных на ней элементов управления.

- `ShortcutMenuBar` — позволяет определить свое собственное пользовательское контекстное меню, которое будет показываться по щелчку правой кнопкой мыши на форме или отчете. Это свойство предусмотрено, кроме формы и отчета, практически для всех элементов управления.

- `Tag` — простой тег для формы. Можно считать его пользовательским атрибутом, который доступен только из программы и ни на что не влияет. Обычно используется для хранения служебной информации (например, меток). Принимает строковое значение с максимальной длиной 2048 символов.

- `TimerInterval` — позволяет установить значение (в миллисекундах), через которое будет срабатывать таймер формы (по умолчанию он отключен). Обычно настраивается в событийной процедуре `Load` для формы. Чаще всего используется с событийной процедурой для события `Timer`. Применяется для организации ожидания (например, мы ждем, пока завершит работу какое-то внешнее приложение, проверяя его состояние каждые полсекунды), для анимации формы и т. п.

- `Toolbar` — позволяет определить панель инструментов, которая будет появляться каждый раз при открытой форме или отчете. Конечно, чаще всего используется своя собственная пользовательская панель инструментов.

- `UniqueTable` — используется тогда, когда таблица привязана к форме или хранимой процедуре, которые ссылаются сразу на несколько таблиц, и нам нужно определить, как в этой ситуации будут вноситься изменения при конфликте.

- `ViewsAllowed` — определяет, в каком режиме пользователь может работать с формой: только в режиме **Form View**, только в режиме **DatasheetView**

или в обоих режимах (по умолчанию). Режим конструктора доступен для пользователя всегда. Запретить его можно только при помощи разрешений (меню **Сервис | Защита | Разрешения**).

- `Visible` — это свойство позволяет скрыть форму или, наоборот, сделать ее видимой.
- `WindowHeight`, `WindowLeft`, `WindowTop`, `WindowWidth` — определяют размеры окна формы.

По сравнению со свойствами методов у объекта `Form` совсем немного. Для большинства из них назначение понятно из названия.

- `GoTo()` — позволяет перейти на указанную страницу многоэкранной формы.
- `Move()` — перемещает форму на экране.
- `Recalc()` — позволяет пересчитать значения в вычисляемых элементах управления формы. Если форма находится в фокусе, то на графическом экране можно вместо этого просто нажать клавишу <F9>.
- `Refresh()` — позволяет отобразить изменения, которые внесены в текущий набор данных в форме. Если нужно еще раз скачать данные из базы (например, они могли быть изменены другими пользователями), то нужно воспользоваться методом `Requery()`.
- `Repaint()` — перерисовывает форму. Обычно используется в тех ситуациях, когда перед этим автоматическая перерисовка была отключена при помощи свойства `Painting`.
- `SetFocus()` — позволяет установить фокус на форме.
- `Undo()` — очищает информацию, которую пользователь ввел для текущей записи (если эта информация ошибочна). Переход на новую запись обычно приводит к сохранению текущей, поэтому после такого перехода метод `Undo()` уже не поможет.

Для формы предусмотрено также несколько десятков событий, которые на практике используются очень активно. Эти события включают в себя как стандартные события форм VBA (например, `Load`), так и специфические, такие как `Query` — запрос к источнику данных, вставка, изменение или удаление записи через форму и т. п. С ними можно работать двумя способами: через специальные свойства формы (которые начинаются на `Before...`, `After...`, `On...` и т. п.) и обычным способом через событийные процедуры. Для этого достаточно в **Project Explorer** в списке объектов выбрать нужную форму, нажать клавишу <F7>, а затем выбрать нужную событийную процедуру (рис. 12.5).

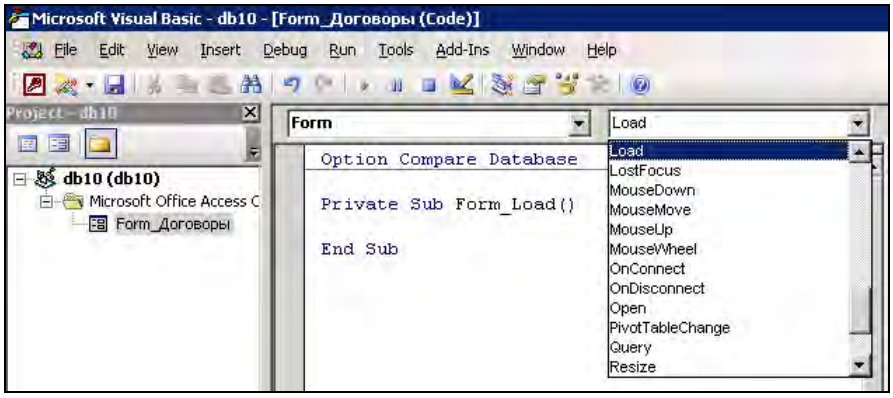


Рис. 12.5. Список событий формы

12.7. Работа с отчетами (объект *Report*)

Еще один часто используемый в программах объект Access — это отчет, представленный объектом `Report`. Отчеты Access — это, возможно, самый простой способ генерации отчетов к базам данных (по сравнению с другими способами генерации отчетов, например, такими, как применение Crystal Reports, Microsoft Reporting Services или уже рассмотренных нами связок VBA/ADO/Word и VBA/ADO/Excel). При помощи отчетов Access можно, конечно, генерировать отчеты не только для самих баз данных Access, но и для внешних источников данных, например, баз данных SQL Server или Oracle. При этом в отчетах дополнительные функциональные возможности (например, условное форматирование) реализуются именно средствами VBA.

С программной точки зрения работа с отчетами очень похожа на работу с формами. Точно так же доступ к объектам всех отчетов можно получить при помощи коллекции `Application.CurrentProject.AllReports` (в которой находятся объекты `AccessObject`), а доступ ко всем открытым отчетам — при помощи коллекции `Reports` с более традиционными объектами `Report`. Точно так же очень редко приходится создавать отчеты программными средствами — обычно эта операция производится из вкладки **Отчеты** окна базы данных. Однако отчеты программным образом приходится создавать все-таки чаще, чем формы. Программно создать отчет можно при помощи метода `Application.CreateReport()`:

```
Dim oReport As Report
Set oReport = Application.CreateReport()
```

В этом случае отчет будет создан только в оперативной памяти, откуда он бесследно исчезнет после завершения работы создавшей его процедуры. Со-

хранить отчет с именем по умолчанию (в русской версии Access это `Отчет1`, `Отчет2` и т. п.) можно, добавив в код строку:

```
DoCmd.Close , , acSaveYes
```

Но, конечно, чаще нам нужно сохранить созданный отчет с указанным нами именем. Для этой цели перед вызовом метода `DoCmd.Close()` нужно поставить вызов другого метода — `DoCmd.Save()`:

```
DoCmd.Save , "Отчет_по_продажам"
DoCmd.Close
```

Метод `CreateReport()` создаст пустой отчет, если только мы не передадим ему в качестве параметра шаблон отчета с уже имеющимися элементами управления. В отчете можно использовать те же элементы управления, что и в форме. Но прежде, чем помещать в отчет элементы управления, необходимо разобраться, где они будут находиться.

Отчет в Access может состоять из девяти областей, из которых по умолчанию видны только три:

- *Верхний колонтитул (Page Header)* — подразумевается верхний колонтитул для страницы. Он будет повторяться столько раз, сколько страниц в отчете. Обычно в него помещается номер страницы, информация о самом отчете, его авторе, в ленточных отчетах (когда в области данных данные идут в виде столбцов) возможно еще и заголовки столбцов;
- *Область данных (Details)*, иногда также называется "подробности" — основная область отчета, в которую обычно выводятся записи из базы данных. Эта область будет повторяться столько раз, сколько записей у нас в базе данных;
- *Нижний колонтитул (Page Footer)* — нижний колонтитул для страницы. Используется обычно для того же, что и верхний колонтитул.

Если щелкнуть правой кнопкой мыши по пустому месту в отчете и в контекстном меню выбрать **Заголовок/примечание отчета**, то будут показаны еще две области отчета (рис. 12.6):

- *Заголовок отчета (Report Header)* — специальная область, которая идет перед всем отчетом и не повторяется. Обычно в нее помещается сводная информация об отчете: автор, количество страниц, время вывода, итоги, диаграмма, представляющая данные в отчете, и т. п.;
- *Примечание отчета (Report Footer)* — область в конец отчета, которая также не повторяется. Используется для тех же целей, что и заголовок отчета.

Если вы используете в отчете группировку, то у вас появятся и другие области — верхние и нижние колонтитулы для ваших групп. Максимальное число

уровней группировки в отчетах Access — два и для каждого из них можно использовать свои колонтитулы. Обычно в них помещаются заголовки групп и итоги по группам.

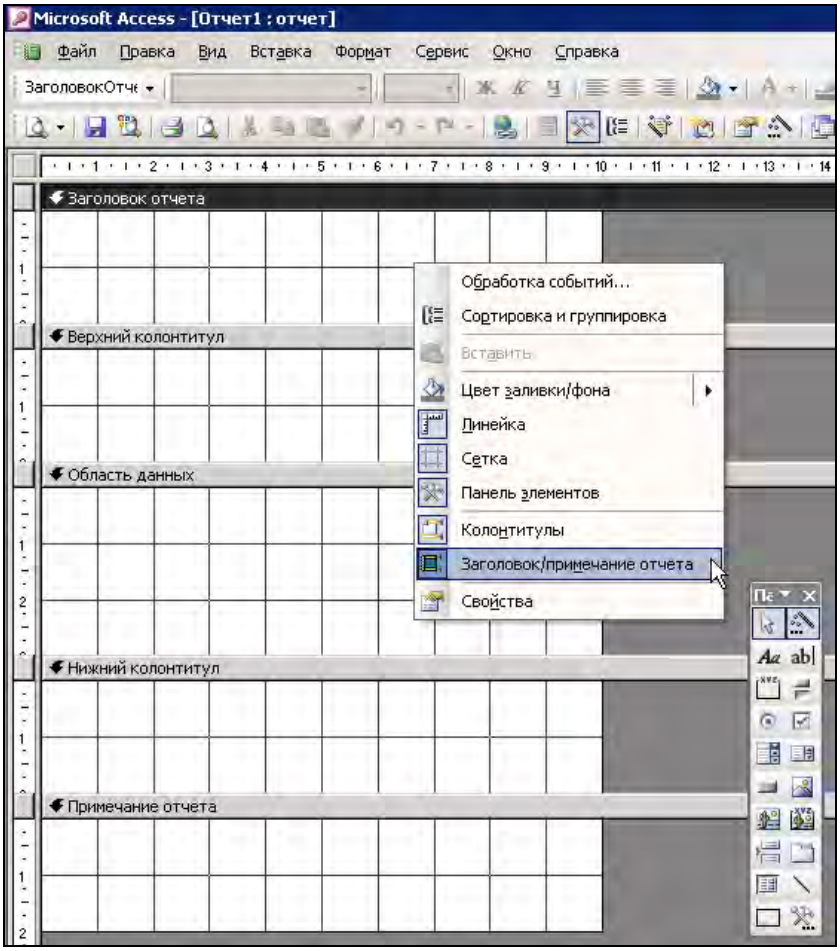


Рис. 12.6. Дополнительные области отчета в окне конструктора

После того, как мы разобрались, куда именно мы хотим поместить элемент управления, для его непосредственного размещения в отчете можно использовать метод `Application.CreateReportControl()`. В качестве параметров этот метод принимает имя отчета (он обязательно должен быть открыт), тип элемента управления, область отчета, в которую необходимо поместить этот элемент управления, а также имя столбца, к которому может быть привязан элемент управления и координаты элемента управления в отчете. Например,

создать в области данных ни к чему не привязанное текстовое поле с именем можно так:

```
Set txt1 = CreateReportControl(oReport.Name, acTextBox, acDetail)
txt1.Name = "txtCustomerID"
```

Если же вы пошли обычным путем и создали отчет не программно, а на графическом экране при помощи мастера или конструктора, то, возможно, вам потребуется его открыть программным образом. Делается это при помощи метода `DoCmd.OpenReport()`. Однако здесь есть одна тонкость: если вы выполните самый простой вариант кода, например:

```
DoCmd.OpenReport "Отчет_по_продажам"
```

то отчет, вместо того, чтобы открыться в окне просмотра, отправится на печать. Чтобы все-таки просмотреть его, нужно использовать код вида:

```
DoCmd.OpenReport "Отчет_по_продажам", acViewPreview
```

Обращаться к элементам управления в отчете можно точно так же, как и к элементам управления формы — при помощи коллекции `Controls`. Например, сделать наш отчет невидимым можно так:

```
oReport.Controls("Отчет_по_продажам").Visible = False
```

Свойства, методы и события объекта `Report` практически полностью совпадают со свойствами, методами и событиями объекта `Form`, которые были рассмотрены в *разд. 12.5*. Единственное принципиальное исключение — в объекте `Report` предусмотрен специальный набор методов для рисования, таких как `Circle()` (нарисовать круг или эллипс), `Line()` (нарисовать линию), `PSet()` (установить цвет для отдельного пиксела) и т. п., а также набор свойств и методов для программного вывода текстовых надписей. Но обычно нет смысла увлекаться довольно трудоемким рисованием или выводом текста, намного проще выполнить необходимые действия в режиме конструктора на графическом экране. Если вместо рисования кругов и линий на экране вы воспользуетесь элементов управления `Image` (рисунок), то возможностей у вас будет намного больше.

12.8. Другие объекты Access

В объектной модели Access предусмотрены и другие объекты, которые в программировании средствами VBA используются реже, чем уже рассмотренные нами `Application`, `DoCmd`, `Form` и `Report`. Далее приведена краткая информация об этих объектах.

В Access предусмотрена очень удобная возможность, которая, по моему опыту, мало известна как программистам, так и пользователям. Эта возможность

называется *Страницы доступа к данным* (*Data Access Pages*). Применение страниц доступа к данным — это самый простой способ создать Web-форму для занесения информации на источник данных (в качестве источников для страниц доступа к данным в настоящее время можно использовать только базы данных Access и Microsoft SQL Server). Физически вы создаете Web-страницу с элементом управления ActiveX, который и обеспечивает необходимую функциональность для подключения к источнику данных, выполнения на нем различных операций и т. п. На форме вы можете использовать привычный набор элементов управления (кнопки, текстовые поля и т. п.), код для которых можно писать на языке VBScript (ближайший родственник VBA). Одним из главных преимуществ страниц доступа к данным является то, что этот код будет выполняться не в среде выполнения браузера, а в среде выполнения этого элемента ActiveX, поэтому в вашем распоряжении останутся все возможности работы с объектными моделями Windows. Например, из кода обработки события `Click` вы можете создать объект `ADO.Recordset`, или запустить Word, Excel или Access на компьютере пользователя, или подключить сетевой диск и т. п.

Страницы доступа к данным можно создавать как в виде объектов базы данных Access (для этой цели в окне базы данных предусмотрена вкладка **Страницы**), так и отдельно от баз данных — просто в виде HTML-файлов. Для этого в Access в меню **Файл** нужно выбрать пункт **Создать**, а потом — **Пустая страница доступа к данным**.

В объектной модели Access страницы доступа к данным представлены объектом `DataAccessPage`. Поскольку такие страницы — это фактически Web-формы, то набор их свойств и методов представляет из себя урезанный набор свойств и методов обычных форм. Web-параметры для страниц доступа данных определяются при помощи специального объекта `WebOptions`.

В формах или отчетах часто приходится использовать условное форматирование, когда формат какого-либо текстового поля или комбинированного списка зависит от различных условий (например, от значения в столбце текущей записи). Например, в зависимости от значения соседнего столбца текстовое поле может становиться то видимым, то невидимым. Для применения условного форматирования используются объекты `FormatCondition`, которые сведены в коллекцию `FormatConditions`.

Объект `GroupLevel` используется при работе с группировкой в отчетах и формах.

Объект `Module` представляет программные модули в базе данных Access — стандартные или модули классов. Эти объекты сведены в коллекцию `Modules`, доступную через одноименное свойство объекта `Application`. Обычно объект `Module` используется для автоматического добавления программного кода в

проекты. Для этой цели в этом объекте предусмотрены специальные методы, такие как `AddFromFile()`, `AddFromString()`, `CreateEventProc()` и т. п.

Для автоматического добавления (или замены) ссылок на объектные библиотеки в Access предусмотрена специальная коллекция `References` с набором объектов `Reference` (ссылок). Свойства и методы коллекции `References` в официальной документации почему-то не рассматриваются, но найти нужные методы можно при помощи подсказок в окне редактора кода. Добавление в проект новой ссылки на объектную библиотеку обычно производится при помощи метода `References.AddFromFile()`.

Объект `SmartTag` представляет смарт-тег — специальное слово или словосочетание, которое автоматически распознается Microsoft Office. В реальных приложениях смарт-теги используются редко, но эта новая возможность приложений Office активно продвигается Microsoft.

Задание для самостоятельной работы 12: Создание приложения VBA в Access

Ситуация:

Приложение, которое было создано для автоматизации формирования договоров в Word *(см. самостоятельную работу к гл. 10)*, решено было изменить, поскольку в нем обнаружились следующие недостатки:

- программа не обеспечивала сохранение данных, введенных пользователем через форму, поэтому эти данные нельзя было использовать повторно. Намного удобнее было бы сохранять вводимые данные в базе данных, например, Access;
- для работы программы необходимы два компонента: программный код (включая форму VBA), который физически находится в файле Normal.dot и шаблон DogovorTemplate.dot, который должен находиться в корневом каталоге диска C:. Такое построение приложения затрудняет его перенос между компьютерами.

ЗАДАНИЕ:

Измените созданное вами в работе 10 приложение, которое обеспечивает автоматическое создание договоров в формате документов Word, таким образом, чтобы оно обеспечивало сохранение данных о договорах в базе данных Microsoft Access. Шаблон документа Word, на основе которого должен формироваться договор, должен находиться в той же базе данных Access. Обеспечьте минимальную защиту приложения от ошибочных действий со стороны пользователя: при запуске приложения пользователь не должен видеть

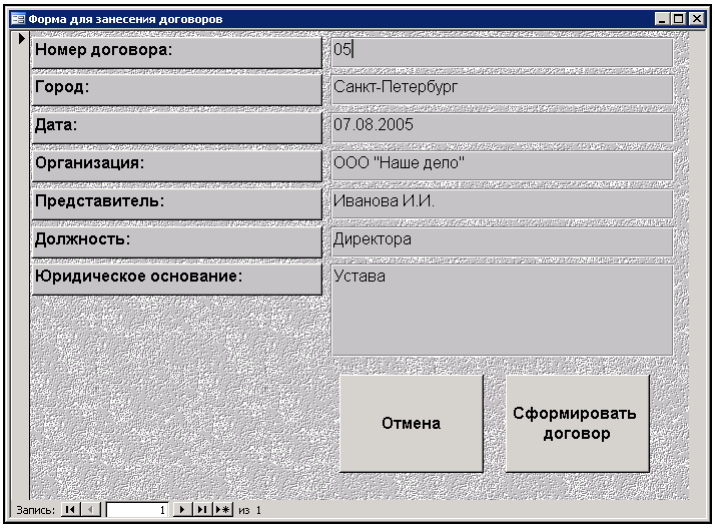


Рис. 12.7. Форма для занесения данных договоров

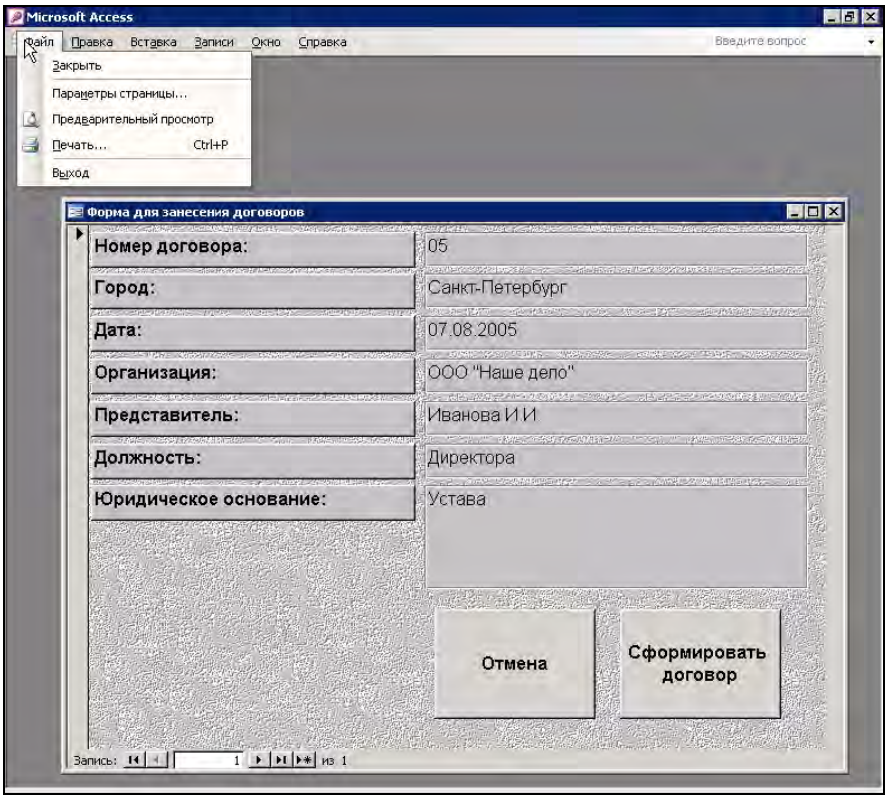


Рис. 12.8. Окно готового приложения

никакие другие объекты, кроме формы для заполнения данных договора. Для этого:

1. Создайте новую базу данных Access, а в ней — необходимые таблицы для хранения данных договоров и шаблонов документов Word.
2. Создайте в этой базе данных форму, аналогичную представленной на рис. 12.7, для занесения информации о договорах в таблицу базы данных и для формирования договора в формате документа Word. Создайте для этой формы необходимый программный код.
3. Обеспечьте минимальную защиту от неверных действий пользователя: при запуске приложения пользователь должен видеть только созданную вами форму (рис. 12.8). Все другие объекты должны быть спрятаны.

Ответ к заданию 12

К пункту 1 задания (создание базы данных и необходимых таблиц):

1. Запустите Microsoft Access и выберите пункт меню **Файл | Создать**. В окне **Создание файла** выберите **Новая база данных**. Сохраните созданную базу данных в корневом каталоге диска C: и назовите ее Dogovors.mdb.
2. В открывшемся окне базы данных перейдите на вкладку **Таблицы** и щелкните два раза левой кнопкой мыши по строке **Создание таблицы в режиме конструктора**. В созданной таблице определите три столбца (рис. 12.9).

   Сохраните эту таблицу как **Шаблоны** и закройте окно конструктора.

Рис. 12.9. Структура таблицы **Шаблоны**

3. В окне базы данных на вкладке **Таблицы** еще раз щелкните по строке **Создание таблицы в режиме конструктора**. Набор столбцов для новой таблицы должен выглядеть так, как приведено в табл. 12.1.

Таблица 12.1. *Структура таблицы* ***Договоры***

Имя столбца	Тип данных	Размер
НомерДоговора	Текстовый (первичный ключ)	10
Город	Текстовый	30

Таблица 12.1 *(окончание)*

Имя столбца	Тип данных	Размер
Дата	Дата/время	50
Организация	Текстовый	50
Представитель	Текстовый	50
Должность	Текстовый	50
ЮрОснование	Текстовый	100

Сохраните эту таблицу с именем **Договоры** и закройте окно конструктора таблицы.

4. На вкладке **Таблицы** окна базы данных щелкните два раза левой кнопкой мыши по созданной таблице **Шаблоны**, чтобы открыть ее в режиме ввода данных. В первую строку этой таблицы в столбец **Описание** введите "Шаблон договора", а затем выделите ячейку в столбце **Шаблон** и в меню **Вставка** выберите **Объект**. В открывшемся окне переставьте переключатель в положение **Создать из файла**, затем нажмите на кнопку **Обзор** и выберите шаблон C:\DogovorTemplate.dot, который вы создали в работе 10. Затем нажмите кнопку **OK**, чтобы поместить шаблон внутрь базы данных.
5. Щелкните по ячейке для помещенного вами шаблона, чтобы убедиться, что он действительно помещен в базу данных. Убедитесь, что в столбце **НомерШаблона** для первой строки автоматически сгенерировано значение 1, закройте эту таблицу.

К пункту 2 задания (создание формы Access и программного кода для формирования файла договора):

1. В окне базы данных перейдите на вкладку **Формы** и щелкните два раза левой кнопкой мыши по строке **Создание формы с помощью мастера**. Откроется окно мастера создания форм.
2. На первом экране мастера в списке **Таблицы и Запросы** выберите **Таблица: Договоры**, затем поместите в список **Выбранные поля** все поля из этой таблицы и нажмите кнопку **Далее**.
3. На следующем экране выберите внешний вид формы в один столбец и нажмите кнопку **Далее**.
4. На следующем экране выберите наиболее вам понравившийся стиль и нажмите кнопку **Далее**.
5. На последнем экране в окне определения имени формы введите имя формы **Форма для занесения договоров**, установите переключатель в поло-

жение **Изменить макет формы** и нажмите кнопку **Готово**. Форма будет открыта в режиме конструктора.

6. Произведите расстановку и поправьте оформление созданных элементов на форме средствами конструктора по вашему вкусу.
7. Добавьте при помощи **Панели инструментов** на свободную часть формы элемент управления **Присоединенная рамка объекта**. Удалите автоматически сгенерированную вместе с ним надпись, а затем откройте свойства этого объекта. Для свойства **Имя** установите значение `OLEObject1`, а для свойства **Вывод на экран** — значение `Нет`.
8. Добавьте на форму две кнопки: **Отмена** и **Сформировать договор**. Элемент управления для первой кнопки должен называться `cmdCancel`, а для второй — `cmdDog`. В открывающемся окне мастера при создании кнопки нажимайте на кнопку **Отмена**.
9. Убедитесь, что для элементов управления текстовых полей оставлены имена по умолчанию (**НомерДоговора**, **Город**, **Дата** и т. п.). В итоге форма в окне конструктора должна выглядеть, например, так, как представлено на рис. 12.10.

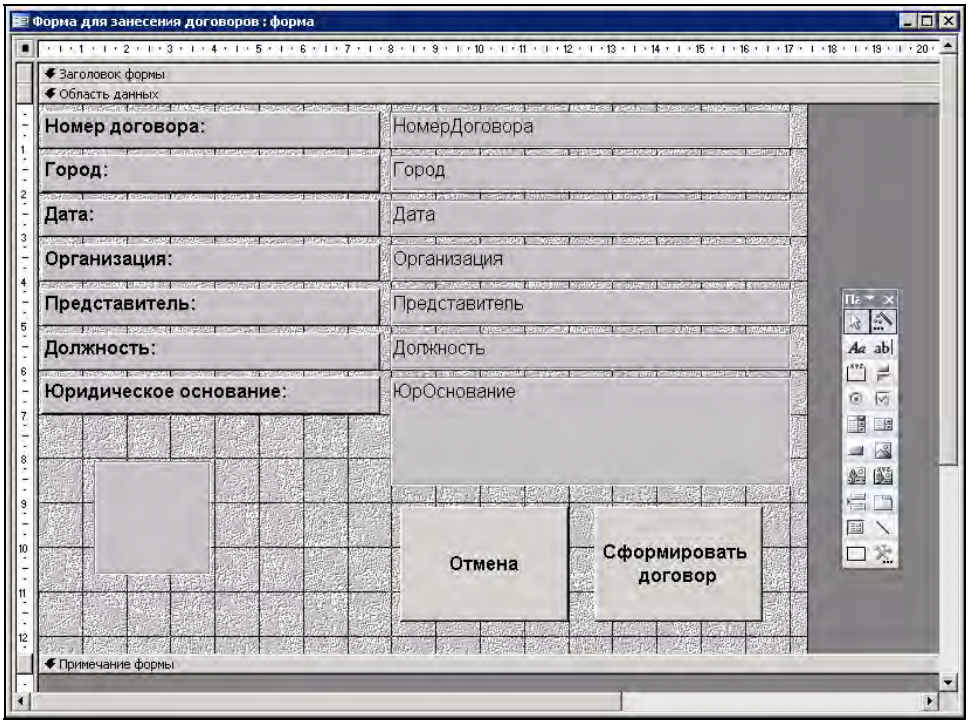


Рис. 12.10. Форма в окне конструктора

10. При помощи меню **Tools | References** в окне редактора кода добавьте ссылку на объектную библиотеку Microsoft Word 11.0 Object Library.
11. Щелкните правой кнопкой мыши по кнопке **Отмена** и в контекстном меню выберите **Обработка событий**. В открывшемся окне **Построитель** выберите **Программы** и нажмите **OK**. Введите следующий код для события `Click` этой кнопки:

```
Private Sub cmdCancel_Click()
    Form.Undo
End Sub
```

12. Так же откройте код для события `Click` кнопки **Сформировать договор** и введите следующий код:

```
Private Sub cmdDog_Click()
    Dim dDate As Date
    Dim НомерДоговора, Город, Организация, Представитель, Должность, _
        ЮрОснование As String

    'Присваиваем значения переменным при помощи элементов управления
    If Form.Controls("Дата").Value <> "" Then _
        dDate = Form.Controls("Дата").Value
    If Form.Controls("НомерДоговора").Value <> "" Then _
        НомерДоговора = Form.Controls("НомерДоговора").Value
    If Form.Controls("Город").Value <> "" Then _
        Город = Form.Controls("Город").Value
    If Form.Controls("Организация").Value <> "" Then _
        Организация = Form.Controls("Организация").Value
    If Form.Controls("Представитель").Value <> "" Then _
        Представитель = Form.Controls("Представитель").Value
    If Form.Controls("Должность").Value <> "" Then _
        Должность = Form.Controls("Должность").Value
    If Form.Controls("ЮрОснование").Value <> "" Then _
        ЮрОснование = Form.Controls("ЮрОснование").Value

    'Получаем шаблон — теперь из базы данных
    Dim oBOF As BoundObjectFrame
    Set oBOF = Form.Controls("OLEObject1")
    oBOF = DLookup("[Шаблон]", "Шаблоны", "[НомерШаблона] = 1")
    oBOF.Verb = acOLEVerbOpen

    oBOF.Action = acOLEActivate

    'Получаем ссылки на запущенный нами Word и открытый в нем документ
    Dim oWord As Word.Application
```

```
    Set oWord = GetObject(, "Word.Application")
    Dim oDoc As Word.Document
    Set oDoc = oWord.ActiveDocument
    oWord.Visible = True
    oWord.ActiveWindow.WindowState = wdWindowStateMaximize
    oDoc.Activate

    'Вставляем данные в закладки
    oDoc.Bookmarks("bNumber").Range.Text = НомерДоговора
    oDoc.Bookmarks("bCity").Range.Text = Город
    oDoc.Bookmarks("bDate").Range.Text = dDate
    oDoc.Bookmarks("bOrg").Range.Text = Организация
    oDoc.Bookmarks("bTitle").Range.Text = Должность
    oDoc.Bookmarks("bPerson").Range.Text = Представитель
    oDoc.Bookmarks("bLaw").Range.Text = ЮрОснование

End Sub
```

13. Запустите созданный вами код на выполнение и убедитесь в его работоспособности.

К пункту 3 задания (обеспечение минимальной защиты от действий пользователя):

1. В окне базы данных Access в меню **Сервис** выберите **Параметры запуска**.
2. В открывшемся окне **Параметры запуска** снимите все флажки, а в списке **Вывод формы/страницы** выберите **Форма для занесения договоров** (рис. 12.11).

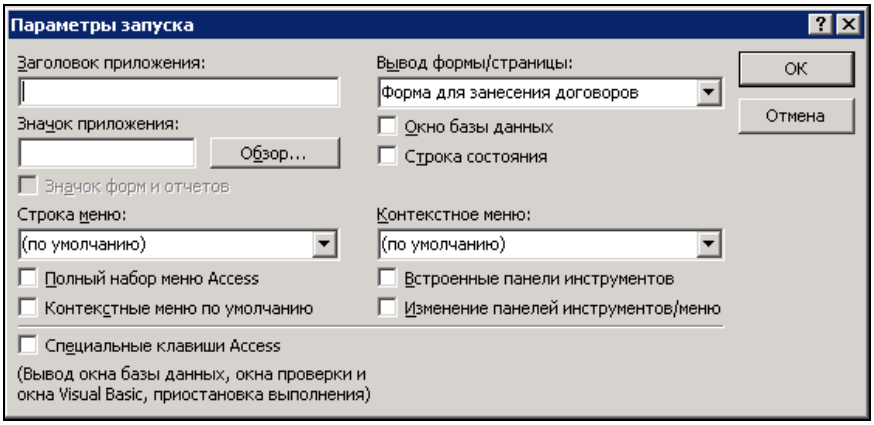


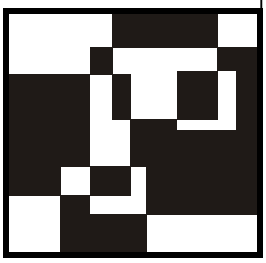
Рис. 12.11. Настройка параметров приложения в окне **Параметры запуска**

3. Нажмите кнопку **OK**. Затем закройте и вновь откройте созданную вами базу данных. Убедитесь, что все объекты базы данных, кроме формы для занесения договоров, спрятаны от пользователя.

Примечание

База данных откроется в обычном режиме, если при ее открытии удерживать нажатой клавишу <Shift>.

ГЛАВА 13



Программирование в Outlook

13.1. Зачем программировать в Outlook

Outlook (вместе с его урезанной версией, которая называется Outlook Express) — самая распространенная программа для работы с электронной почтой. Однако его важность заключается не только в возможности отправки и получения электронных сообщений. На предприятиях очень большую ценность представляют его дополнительные возможности, которые помогают делать то, что Microsoft называет задачами персонального информационного менеджера (Personal Information Manager, PIM).

Первая такая задача — это *работа с календарем*, т. е. организация времени пользователя. **Календарь** тесно интегрирован с другими элементами Outlook (такими как **Контакт** и **Задача**), а также с внешними приложениями (например, Microsoft Project). На предприятиях часто используется назначение задач пользователям, когда такие задачи автоматически появляются как элементы **Календаря**. Если пользователь самостоятельно заносит свои задачи в Outlook, то можно (если почтовый ящик пользователя лежит на сервере Exchange Server) предоставить доступ к календарю этого пользователя его менеджерам. Менеджеры смогут видеть, какие задания есть в настоящий момент у этого пользователя, что позволит избежать ситуаций, когда новое задание выдается сотруднику, уже занятому важной работой.

У **Календаря** есть еще одна замечательная возможность: если вы создадите из Outlook на сервере Exchange общую папку с элементами типа **Календарь**, в вашем распоряжении будет готовое приложение для планирования совместного доступа к ресурсам (комнатам для переговоров, проекторам, разному оборудованию и т. п.). Предположим, например, что в вашей организации принято правило: все личное общение с клиентами должны производиться только в комнате для переговоров, а количество этих комнат ограничено.

Вполне может произойти ситуация, когда клиент придет, а все комнаты будут заняты. Чтобы этого не произошло, используется общая папка с элементами управления типа **Календарь**. В процессе общения с клиентом пользователь открывает из Outlook эту общую папку и смотрит, в какое время комната свободна. Согласовав с клиентом время визита, он создает в этой общей папке элемент типа **Встреча**, и уже другие сотрудники смогут увидеть, что в это время комната для переговоров занята.

Конечно же, на практике как синхронизация календаря с другими источниками, так и использование общих календарей часто требует добавления функциональности средствами программирования.

Вторая задача — это *работа с контактами*. Контакты Outlook — это наиболее распространенный и стандартный формат адресной книги. В последнее время особенно важным стало то, что контакты из адресной книги Outlook можно очень легко и просто синхронизировать с сотовыми телефонами, карманными компьютерами и прочими аналогичными устройствами. На предприятиях часто используется, например, общий список контактов для подразделения в виде элементов **Контакт**, которые хранятся в общей папке Exchange Server (адресная или телефонная книга самой организации также обычно ведется в формате адресной книги Exchange Server и доступна через Outlook). Ситуаций, когда вам нужно программным образом создать контакты (например, на основе информации из базы данных), или синхронизировать их, или пройти циклом по всем контактами и изменить их в зависимости от какого-то условия, очень много.

Третья возможность — это *работа с задачами и поручениями*. Для масштабных проектов, конечно, лучше использовать специализированное программное обеспечение (например, Microsoft Project и Project Server), но для простых проектов, за которые ответственен один менеджер, задачи Outlook вполне подойдут. При помощи этого средства можно создавать задачи, назначать их другим лицам (поручения) с уведомлением их по электронной почте, отслеживать процент выполнения и т. п.

Четвертая задача — это *работа с дневником*. При помощи этого средства вы можете отслеживать работу с документами Office, обмен информацией при помощи Outlook и т. п. Дневник позволяет создать отчет о проделанной работе или вспомнить, в каком файле находится нужный документ за прошлый квартал, чтобы по его образцу сделать новый документ.

Есть еще *работа с заметками* — компьютерными аналогами маленьких листочков с напоминаниями, которыми некоторые любители обклеивают все вокруг себя.

Конечно же, перечисленными встроенными возможностями работа с Outlook не ограничивается. На связке Outlook—Exchange Server основана целая об-

ласть программирования, которая называется collaboration development — *разработка приложений коллективного использования*. Основные задачи, которые решаются при помощи приложений коллективного использования, — это сбор и автоматизированная обработка внутрикорпоративной информации.

Например, представим себе следующую задачу из реальной жизни: каждый банк должен в конце каждого месяца представлять в Центральный банк информацию об экономических нормативах. Чаще всего это выглядит так: сотрудник планово-экономического отдела, ответственный за сбор информации, в начале каждого месяца идет в бухгалтерию, чтобы получить информацию об остатках на требуемых счетах на конец прошлого месяца. Затем он отправляется в кредитный отдел, чтобы получить информацию о том, какие кредиты относятся к какой категории. После этого он руками формирует файл отчета требуемого формата. При использовании средств Outlook это могло бы выглядеть по-другому:

1. В начале месяца у сотрудника планово-экономического отдела в папке **Входящие** в Outlook автоматически появляется специальная форма для заполнения информации о нормативах.
2. Он нажимает кнопку **Отправить в бухгалтерию** на этой форме, и она отправляется ответственному сотруднику бухгалтерии (он может и автоматически получать информацию об остатках на счетах из программы **Операционный день**, если разработчик сможет это реализовать).
3. Сотрудник бухгалтерии, заполнив нужные поля, нажимает кнопку **Дальше** и форма с сохраненными данными идет в кредитный отдел.
4. Сотрудник кредитного отдела заполняет свои поля (поля, заполненные в бухгалтерии, при этом автоматически должны быть доступны только для чтения) и нажимает кнопку **Дальше**.
5. Форма со всеми необходимыми данными приходит к сотруднику планово-экономического отдела, тот нажимает на ней кнопку **Сформировать отчет**, и формируется файл отчета в нужном формате, а форма с сохраненными значениями автоматически помещается в архив.

На любом предприятии задач по сбору внутрикорпоративной информации очень много. Можно привести в пример и отчеты о командировках, сбор информации из филиалов, формы, заполняемые продавцами (в офисе и в командировках), информацию инвентаризаций — этот перечень можно продолжать бесконечно. И, как показывает практика, самый удобный способ — это именно применение средств Outlook и Exchange Server. Большим плюсом здесь является то, что никаких внешних средств разработки использовать не нужно: Outlook — это еще и среда разработки. Чаще всего в подобных приложе-

ниях используются формы Outlook — специальные шаблоны сообщений с элементами управления и программными возможностями, например, маршрутами прохождения (эти формы ни имеют никакого отношения к обычным формам VBA или формам Access). Создать такую форму и определить для нее необходимый программный код можно средствами самого Outlook: для этого достаточно в меню **Сервис | Формы** воспользоваться пунктом **Конструктор форм**, выбрать нужную форму (например, **Сообщение** в **Библиотеке стандартных форм**) и нажать кнопку **Открыть**. Откроется окно дизайнера форм (рис. 13.1), в котором вы сможете изменять шаблон стандартного сообщения, как вам угодно: помещать новые элементы управления, привязывать к ним программный код (при помощи меню **Форма | Просмотреть код**) и т. п.

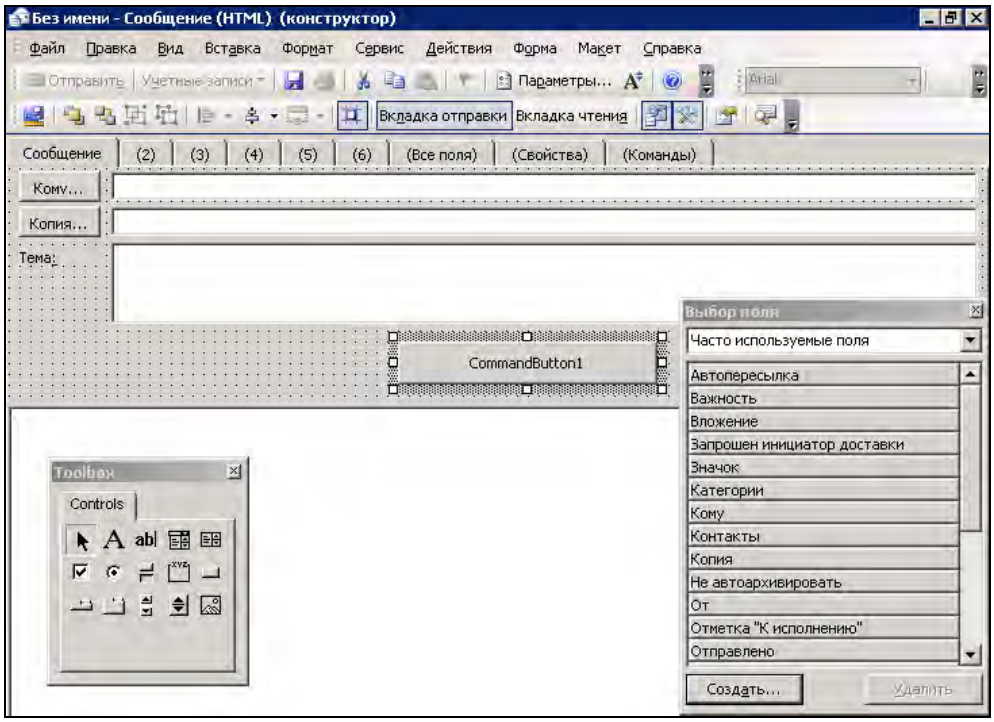


Рис. 13.1. Окно конструктора форм Outlook

Например, добавить код для нашей кнопки `CommandButton1`, которую мы поместили на форму, можно так:

1. В меню **Форма** выберите **Просмотреть код** (вместо этого можно воспользоваться кнопкой **Просмотреть код** на панели инструментов **Конструктор форм**);

2. В окне редактора сценариев добавьте следующий код:

```
Function CommandButton1_Click()
    MsgBox "Привет из формы Outlook"
End Function
```

   К сожалению, автоматически сгенерировать событийную процедуру для кнопки не получится. Нет в нашем распоряжении и подсказок по свойствам и методам, и подсветки синтаксиса, и отладки, и многого другого.

3. Чтобы запустить кнопку на выполнение, воспользуйтесь командой **Выполнить форму** в меню **Форма**, а затем нажмите кнопку.

Как вы уже, наверное, догадались, это не совсем привычный нам VBA (и даже больше — это вообще другой язык программирования VBScript). Для форм Outlook предусмотрена своя собственная среда программирования, своя объектная и событийная модели. Как правило, работа с формами Outlook неотделима от работы с корпоративными возможностями Exchange Server: библиотеками форм, серверными скриптами, общими папками, маршрутизацией и т. п. Все это очень большая специальная тема, для рассмотрения которой потребуется отдельная толстая книга. По этой причине работу с формами Outlook и приложениями коллективного использования здесь мы рассматривать не будем. Скажем только, что хорошей отправной точкой для самостоятельного освоения этой темы может послужить файл официальной документации Microsoft, который по умолчанию находится в каталоге C:\Program Files\Microsoft Office\OFFICE11\1049\OLFM10.CHM, и сайт **www.slipstick.com**. В этой главе мы сосредоточимся на работе в Outlook традиционными средствами VBA, при помощи стандартных модулей и форм VBA и в привычном редакторе кода.

Задач автоматизации, которые могут решаться средствами VBA, также очень много:

- организация рассылки электронной почты по расписанию;
- получение электронной почты и ее автоматизированная обработка (например, если сообщение пришло в стандартном формате, можно извлечь из него данные и поместить в базу данных), например, сбор данных из филиалов;
- автоматизация работы с контактами, например, импорт контактов из базы данных или файла Excel в общую папку на сервере Exchange;
- автоматическое создание или изменение записей в календаре;
- и многое-многое другое.

Работать с этими возможностями Outlook нам и предстоит научиться.

13.2. Некоторые особенности программирования в Outlook

Программирование в Outlook имеет ряд интересных особенностей, о которых необходимо упомянуть.

Первая особенность заключается в том, где именно хранятся программные модули Outlook, в которых мы создаем код. Как мы помним, в Word они хранятся вместе с документами (или шаблонами, например, Normal.dot), в Excel — в файлах рабочих книг, в Access — в файлах баз данных MDB. В Outlook информация стандартных модулей хранится в файле личных папок PST, который по умолчанию создается в профиле данного пользователя. В результате, с одной стороны, работа с программным кодом VBA в Outlook упрощается: для данного пользователя на этом компьютере он становится доступен из Outlook всегда. С другой стороны, становится труднее предоставить этот код в распоряжение другого пользователя. В этой ситуации можно использовать два выхода:

- первый выход — *воспользоваться средствами экспорта и импорта программных модулей*, которые доступны из контекстного меню для модуля в **Project Explorer** (рис. 13.2);

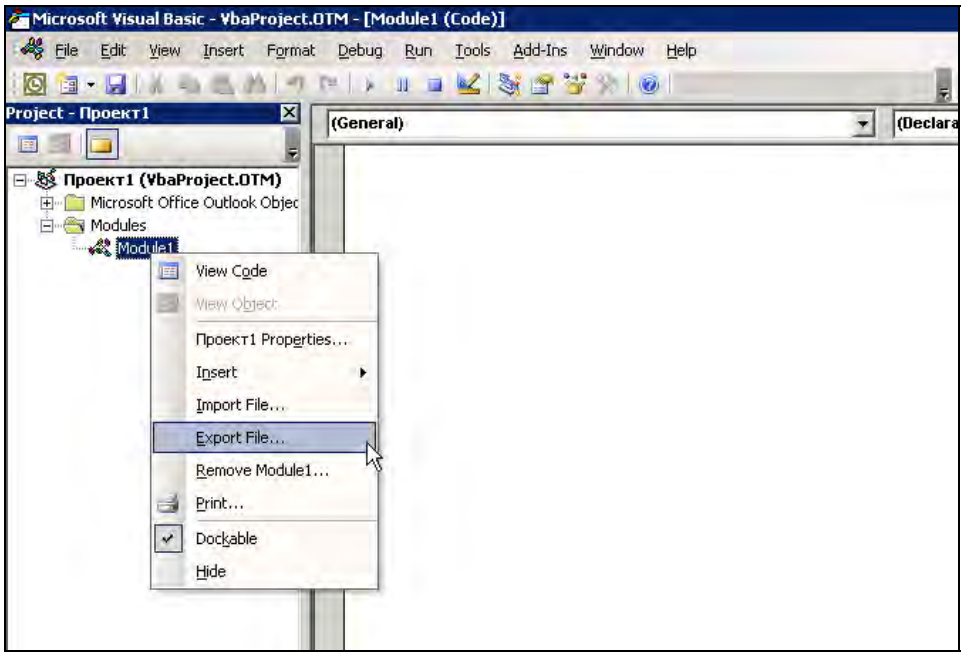


Рис. 13.2. Средства импорта и экспорта программного кода

- второй выход — *создать контейнерное приложение*, например, файл Word или книгу Outlook, из которого программным образом запускать Outlook и выполнять в нем необходимые действия.

Вторая особенность работы с VBA в Outlook заключается в том, что в Outlook реализована концепция пространства имен. Пространство имен в Outlook формально определяется как абстрактный корневой объект для любого источника данных (например, папки в почтовом ящике Exchange или PST-файле на диске). Проще всего представить себе пространство имен Outlook как некий драйвер, который нужно использовать для подключения к данным. В настоящее время Outlook поддерживает только одно пространство имен — MAPI (Messaging Application Programming Interface, интерфейс прикладного программирования для работы с сообщениями), но разработчики Outlook требуют, чтобы это пространство имен явно указывалось при выполнении самых разных операций. Подробнее про объект `Namespace`, представляющий пространство имен, будет рассказано в *разд. 13.4*. Например, для запуска Outlook и открытия в нем папки **Inbox** (Входящие) из другой программы придется использовать следующий код (не забудьте подключить ссылку на библиотеку Microsoft Outlook 11.0 Object Library):

```
Dim oOutlook As New Outlook.Application
Set oNameSpace = oOutlook.GetNamespace("MAPI")
Set oInbox = oNameSpace.GetDefaultFolder(olFolderInbox)
oInbox.Display
```

Третья особенность работы с Outlook заключается в некоторой терминологической путанице. Как правило, в документации по объектным моделям приложений Office термин `Item` (элемент) применяется к элементам коллекций. В Outlook он получает второе значение: `Item` — это все, что может храниться в папках Outlook: почтовые сообщения (объект `MailItem`), контакты (объект `ContactItem`), встречи (объект `Appointment`) и т. п. Не путайте!

У Outlook есть еще одна особенность. На протяжении многих лет Outlook была программой, которая первой подвергалась атакам вирусов, троянских программ и прочего вредоносного программного обеспечения, приходящего по электронной почте. Иногда такие атаки бывали успешными, и Outlook на компьютере пользователя сам начинал рассылать электронные письма с вирусами (в том числе и при помощи кода VBA). Чтобы снизить вероятность развития событий по такому сценарию, разработчики Outlook сознательно поместили в его объектную модель ограничения, которые должны препятствовать распространению вирусов. Иногда эти ограничения (у них есть специальное название — *Outlook Object Model Guard*) мешают и нормальной работе программ VBA. О них будет рассказано в следующих разделах. Иногда по причине таких ограничений бывает удобнее использовать вместо объектной

модели Outlook библиотеку CDO, которая имеется на любом компьютере с Windows 2000, XP или 2003.

Интересной особенностью Outlook является и то, что в отличие от других приложений Office вы не сможете напрямую (при помощи ключевого слова `New` или команды `CreateObject()`) создать ни одного объекта Outlook, кроме `Application`. Для создания всех остальных объектов придется использовать соответствующие методы уже созданных объектов.

На всякий случай также заметим, что макрорекордера в Outlook, как и в Access, к сожалению, нет. Всю необходимую дополнительную информацию вам придется искать при помощи документации.

13.3. Объект *Application*, его свойства и методы

Как и у всех приложений Office, на вершине объектной модели Outlook находится объект `Application`. Его можно использовать для запуска Outlook из внешних приложений (см. пример в предыдущем разделе). Отличительной особенностью объекта `Application` в Outlook является относительно небольшое количество свойств и методов (за счет того, что часть из них переехала в объект `Namespace`). Наиболее часто используемые свойства объекта `Application` представлены далее.

- `Explorers` — это свойство возвращает коллекцию `Explorers` с объектами `Explorer`, каждый из которых представляет собой папку Outlook, открытую на просмотр пользователем. Основное назначение этой коллекции и объектов `Explorer` — проверка, не открыта ли уже пользователем в Outlook та или иная папка, и, в зависимости от результата, активизация этого окна (`Explorer.Activate()`) или его закрытие (`Explorer.Close()`). Метод `ActiveExplorer()` объекта `Application` позволяет получить ссылку на окно, активное в настоящий момент, а `GetExplorer()` — получить ссылку на объект `Explorer` для указанной папки Outlook (без его автоматической активизации).
- `Inspectors` — свойство, которое очень похоже на `Explorers`. Оно возвращает коллекцию `Inspectors` с объектами `Inspector`. Главное отличие в том, что объекты `Inspector` представляют не открытые папки Outlook, как `Explorer`, а открытые на просмотр и редактирование элементы (например, почтовые сообщения). Объект `Inspector` используется для таких же проверок, что и объект `Explorer`, его свойства и методы почти полностью совпадают со свойствами и методами объекта `Explorer`. Для тех же целей предусмотрены и методы `ActiveInspector()` и `GetInspector()` объекта `Application`.

- `Reminders` — позволяет вернуть коллекцию `Reminders` с объектами `Reminder`, представляющими текущие оповещения. Обычно это свойство используется для того, чтобы программным образом отключить все оповещения:

```
Dim oOutlook As New Outlook.Application
Dim oReminder As Outlook.Reminder
For Each oReminder In oOutlook.Reminders
    oReminder.Dismiss
Next
```

- `Session` — это свойство позволяет вернуть объект `Namespace`, представляющий пространство имен для текущего сеанса (т. е. пространство имен MAPI). Это свойство можно использовать вместо метода `GetNamespace()`. Подробнее про объект `Namespace` будет рассказано в *разд. 13.4*.

  Аналогичное свойство `Session` предусмотрено для самого объекта `Namespace` и для множества других объектов Outlook.

Теперь расскажем о методах объекта `Outlook.Application`.

- Методы с префиксом `Active...` — просто возвращают ссылку на активный в настоящее время объект `Explorer` или `Inspector`.
- `AdvancedSearch()` — очень важный метод. Он позволяет производить поиск по папкам Outlook (что на практике требуется достаточно часто). Подробнее про этот метод и сопутствующие ему объекты `Search` и `Results` будет рассказано в *разд. 13.7*.
- `CopyFile()` — позволяет скопировать файл с диска в папку Outlook. Можно использовать, например, для переноса всех файлов из каталога с документацией по проекту в общую папку Exchange Server или в библиотеку документов SharePoint Portal Server.
- `CreateItem()` — метод, который используется очень часто. Он позволяет создать новые элементы в Outlook. Например, создать новый элемент типа контакт, заполнить его свойства, сохранить, а затем открыть для просмотра можно так:

```
Dim oOutlook As New Outlook.Application
Dim oContact As Outlook.ContactItem
Set oContact = oOutlook.CreateItem(olContactItem)
oContact.FirstName = "Академия специальных курсов"
oContact.Email1Address = "info@askit.ru"
oContact.Save
oContact.Display
```

А теперь представьте, что вы создаете объекты контактов в цикле на основе записей из базы данных или строк в таблице Excel. Справочник контактов будет загружен в Outlook очень быстро и эффективно. Только не забывайте после каждого создания и сохранения контакта удалять его объект из оперативной памяти — иначе память на компьютере кончится и это приведет к ошибке. В нашем примере удалить объект контакта из памяти можно при помощи строки:

```
Set oContact = Nothing
```

- `CreateItemFromTemplate()` — точно так же создает новый элемент Outlook, но уже на основе шаблона Outlook в файловой системе — файла oft.
- `GetNameSpace()` — метод, который используется, наверное, в большинстве программ VBA в Outlook. Позволяет получить объект пространства имен MAPI. Подробнее про работу с этим объектом будет рассказано в следующем разделе.
- `IsSearchSynchronous()` — используется для проверки режима поиска *(см. разд. 13.7)*.
- `Quit()` — осуществляет выход из Outlook.

13.4. Объект *Namespace*

Как уже говорилось ранее, в Outlook используется понятие пространств имен — нечто вроде драйверов, которые, по замыслу разработчиков Outlook, должны обеспечивать доступ к различным хранилищам пользовательских данных. У каждого такого драйвера должны быть свои возможности. Однако уже на протяжении долгого времени в Outlook используется единственное пространство имен — MAPI, и пока нет никакой информации о том, что должно появиться что-то еще. Для наших целей пространство имен Outlook можно рассматривать просто как специальный служебный объект, в который "переехали" некоторые свойства и методы объекта `Application`.

Если в программе вам нужны свойства или методы объекта `Namespace`, получить ссылку на этот объект можно двумя способами:

1. Воспользоваться методом `GetNameSpace()` объекта `Application`:

   ```
   Set oNameSpace = Application.GetNamespace("MAPI")
   ```

2. Воспользоваться свойством `Session` того же объекта `Application`:

   ```
   Set oNameSpace = Application.Session
   ```

Эти две строки кода по функциональности полностью равнозначны, но в документации обычно всегда используется только первый способ.

Объект `Namespace` нужен для выполнения самых распространенных операций с электронной почтой: установка соединения с сервером электронной почты, отправка и получение электронной почты, выбор нужной папки, работа с адресными книгами и многих других. Кроме того, объект `Namespace` представляет еще и виртуальный корень папок Exchange, при помощи которого можно циклом проходить по всем папкам в Exchange.

В практической работе объект `Namespace` используется очень часто. Предположим, что вам нужно из Outlook скачать почтовые сообщения для всех учетных записей электронной почты и что-то сделать с каждым полученным сообщением. В нашем примере мы будем просто выводить тему каждого сообщения. На практике можно "разбирать" каждое сообщение в стандартном формате и помещать из него информацию в базу данных. Такое решение может пригодиться при обработке информации, получаемой из филиалов, с Web-сайта предприятия, который находится у провайдера, от торговых представителей, которые находятся в командировке, и т. п.

Как может выглядеть подобное сообщение?

Представим себе, что мы работаем из внешнего приложения. Удобнее всего нам будет использовать форму Word или Excel, чтобы работа с событиями производилась из окна редактора кода. Первое, что нужно сделать, — это поместить в проект ссылку на объектную библиотеку Microsoft Outlook 11.0 Object Library.

Затем мы занимаемся привычным делом — создаем самую обычную форму VBA и помещаем в нее кнопку `CommandButton1`. Перед созданием кода для этой кнопки необходимо сделать еще одно дело: зарегистрировать события для некоего абстрактного объекта `Outlook.Items` в разделе **General—Declarations** редактора кода. Это можно сделать при помощи строки:

```
Public WithEvents oItems As Outlook.Items
```

После этого объект `oItems` с событиями появится в окне редактора кода.

Затем можно создавать код для события `Click()` нашей кнопки. Он может быть, например, таким:

```
Private Sub CommandButton1_Click()

    'Запускаем Outlook
    Dim oOutlook As New Outlook.Application

    'Очень важно: указываем, что события oItems — это события
    'папки Inbox в Outlook
    Set oItems = _
      oOutlook.GetNamespace("MAPI").GetDefaultFolder(olFolderInbox).Items
```

```
    'Дальше просто запускаем прием почты со всех учетных записей
    Dim oNamespace As Outlook.NameSpace
    Set oNamespace = oOutlook.GetNamespace("MAPI")
    oNamespace.SyncObjects("Все учетные записи").Start

End Sub

'Ловим события появления нового сообщения
Private Sub oItems_ItemAdd(ByVal Item As Object)
    MsgBox Item.Subject
End Sub
```

В документации Microsoft для события `ItemAdd` указано, что оно может не срабатывать, если в папку одновременно добавляется большое количество элементов, но у меня он всегда срабатывал правильно.

Конечно, если у вас на предприятии установлен Exchange Server, то для обработки всей входящей электронной почты правильнее использовать серверные скрипты Exchange и его событийную модель. Но реальная жизнь, как всегда, сложнее. Например, часто за Exchange Server отвечает администратор сети, который совершенно не собирается разрешать разработчикам настраивать на нем какие-то скрипты. Другой случай — когда на вашем предприятии работает не Exchange, а, к примеру, почтовая система на UNIX — SendMail, PostFix, CommunigatePro и т. п. Чтобы разобраться с их событиями, может потребоваться много времени и усилий, а Outlook всегда под рукой.

А теперь, как обычно, рассмотрим информацию о самых важных свойствах и методах объекта `Namespace`.

- `AddressLists` — это свойство возвращает коллекцию `AddressLists`, в которой находятся объекты `AddressList`, представляющие все адресные книги, доступные в настоящий момент. Например, получить список всех доступных в данный момент для пользователя адресных книг можно так (подразумевается, что вы работаете с Outlook из внешнего приложения):

  ```
  Dim oOutlook As New Outlook.Application
  Dim oNameSpace As Outlook.NameSpace
  Dim oAddress As Outlook.AddressList
  Set oNameSpace = oOutlook.GetNamespace("MAPI")
  For Each oAddress In oNameSpace.AddressLists
      Debug.Print oAddress.Name
  Next
  ```

  В объекте `AddressList` находится коллекция `AddressEntries` с объектами `AddressEntry`, представляющими записи в адресных книгах. При помощи этой объектной "ветви" вы можете программным способом добавлять записи в адресную книгу, удалять их, изменять свойства и т. п.

Если Outlook у вас работает сам по себе, в работе с коллекцией `AddressLists` нет никакого смысла — единственной доступной для пользователя адресной книгой будет являться книга **Контакты**, с которой проще работать другим способом (при помощи объектов `ContactItem`). Но если Outlook подключен к Exchange Server, то эта возможность становится очень интересной.

- `CurrentUser` — еще одно очень полезное свойство. Возвращает информацию о текущем пользователе (от имени которого открыт Outlook) в виде объекта `Recipient`, при помощи которого можно получить, например, адрес электронной почты данного пользователя, его имя и прочие атрибуты, которые предусмотрены для записи в списке адресов (если они были определены для пользователя). Например, получить доступ к информации об адресе электронной почты текущего пользователя можно так:

```
Set oNameSpace = Application.GetNamespace("MAPI")
Set oRecipient = oNameSpace.CurrentUser
Debug.Print oRecipient.Address
```

  К сожалению, попытка выполнить этот код из внешнего приложения приведет к появлению окна сообщения с вопросом: хотите ли вы предоставить программе доступ к адресам электронной почты в Outlook? Скорее всего, это окно вам совершенно не нужно, но оно было сделано специально в целях безопасности и избежать его вам не удастся. В вашем распоряжении два варианта: программно имитировать нажатие клавиш <Shift>+<Tab> и <Enter> в этом окне (заботливые разработчики отключили даже горячую клавишу для кнопки **Да**) или запускать этот код из уже работающего Outlook, тогда предупреждения не возникнет. Для программной имитации нажатий клавиш можно использовать объект `WshShell` объектной библиотеки Windows Script Host, но поскольку при выполнении кода VBA ошибки не происходит, а просто "подвисает" последняя строка `Debug.Print`, то возникают дополнительные сложности. Их можно обойти только средствами Windows API.

- `ExchangeConnectionMode` — очень важное для практической работы свойство. Оно позволяет определить, настроен ли Outlook для работы с Exchange Server и подключен ли он к Exchange Server в настоящий момент.

- `Folders` — это, наверное, самое главное свойство объекта `Namespace`. Возвращает коллекцию `Folders` с объектами `MAPIFolder`, представляющими папки верхнего уровня в Outlook (у каждой папки, в свою очередь, есть свое свойство `Folders`, так что вы вполне можете пройти циклом по всем без исключения папкам Outlook). Кроме того, что у папок Outlook есть множество своих собственных важных свойств и методов, через свойство `Items` папки вы можете получить доступ ко всем элементам папки (сооб-

щениям, контактам, элементам календаря и т. п.). Подробнее про коллекцию `Folders` и объект `MAPIFolder` будет рассказано в следующем разделе.

- `Offline` — позволяет выяснить, подключен ли в настоящее время Outlook к серверу электронной почты или нет.
- `SyncObjects` — возвращает одноименную коллекцию с объектами `SyncObject` (объекты синхронизации). Сами эти объекты представляют собой группы отправки (то, что при помощи графического интерфейса Outlook можно найти в меню **Сервис | Отправить/Получить**). Самое важное, что можно сделать при помощи объектов `SyncObject`, — это программно инициировать соединение с сервером электронной почты или разорвать его.

Обычно в приложениях, использующих объектную модель Outlook, без методов объекта `Namespace` также не обойтись.

- `AddStore()` и `AddStoreEx()` — позволяют программно открыть файл PST хранилища сообщений Outlook на диске. Обратите внимание на то, что если такого файла на диске нет, то при вызове этого метода Outlook просто создаст его. `AddStoreEx()` отличается тем, что позволяет указать кодировку для файла сообщений. Закрыть PST-файл можно при помощи метода `RemoveStore()`.
- `CreateRecipient()` — позволяет программным образом создать объект `Recipient`. Обычно он используется для передачи в виде параметра при вызове метода `GetSharedDefaultFolder()`. Этот метод необходим для подключения к папке в чужом почтовом ящике.
- `Dial()` — позволяет открыть диалоговое окно **Новый звонок**, чтобы пользователь мог установить коммутируемое соединение. В качестве необязательного параметра принимает имя контакта, который будет автоматически подставлен в это окно.
- `GetDefaultFolder()` — важнейший метод, возвращающий объект `MAPIFolder` для одной из двенадцати встроенных (используемых по умолчанию) папок Outlook: **Входящие**, **Контакты**, **Календарь**, **Отправленные** и т. п. Например, подключиться к папке **Контакты** (и для наглядности открыть ее) можно так:

```
Dim oOutlook As New Outlook.Application
Set oNameSpace = oOutlook.GetNamespace("MAPI")
Set oInbox = oNameSpace.GetDefaultFolder(olFolderContacts)
oInbox.Display
```

- Методы `Get...FromID()` — обычно используются только тогда, когда вы переводите свой код, написанный с использованием объектной библиоте-

ки CDO (о ней — в *разд. 13.8*), на использование объектной модели Outlook.

- `GetSharedDefaultFolder()` — этот метод делает то же, что и `GetDefaultFolder()`, но применяется тогда, когда у пользователя в Outlook кроме своего ящика открыты еще и почтовые ящики других пользователей. Метод позволяет получить ссылку, например, на папку **Inbox** в другом почтовом ящике.
- `Logon()` — позволяет установить соединение по протоколу MAPI (т. е. установить соединение с Exchange Server). В качестве необязательных параметров принимает имя почтового профиля, параметр, определяющий, показывать ли пользователю диалоговое окно выбора профиля и т. п. Разрыв соединения производится при помощи метода `Logoff()`.
- `PickFolder()` — открывает диалоговое окно **Выбор папки**. Если пользователь выбрал папку в этом окне и нажал **OK**, то возвращается объект `MAPIFolder` для выбранной папки.

13.5. Коллекция *Folders* и объект *MAPIFolder*

Обычно, когда мы программным образом работаем с Outlook, нам нужно что-то сделать с его элементами — почтовыми сообщениями, контактами, встречами в календаре и т. п. Все эти элементы расположены в папках Outlook (либо встроенных, либо созданных пользователем). Папкам в объектной модели Outlook соответствуют объекты `MAPIFolder`, которые сведены в коллекцию `Folders`.

В Outlook папки могут быть вложены друг в друга. На самом верху расположены папки верхнего уровня (это не **Входящие**, **Контакты**, **Черновики** и т. п., как вы могли подумать, а папки более высокого уровня, например, **Личные папки**, **Общие папки**, **Mailbox — Administrator**). Доступ к коллекции `Folders`, представляющей собой папки самого верхнего уровня, производится через свойство `Folders` объекта `Namespace` *(см. разд. 13.4)*:

```
Dim oOutlook As New Outlook.Application
Dim oNameSpace As Outlook.NameSpace
Dim oFolder As Outlook.MAPIFolder
Set oNameSpace = oOutlook.GetNamespace("MAPI")
For Each oFolder In oNameSpace.Folders
    Debug.Print oFolder.Name
Next
```

Если вам нужна конкретная встроенная папка, например **Inbox**, то проще всего найти ее при помощи метода `GetDefaultFolder()` объекта `Namespace`:

```
Dim oOutlook As New Outlook.Application
Dim oNameSpace As Outlook.NameSpace
Dim oFolder As Outlook.MAPIFolder
Set oNameSpace = oOutlook.GetNamespace("MAPI")
Set oFolder = oNameSpace.GetDefaultFolder(olFolderInbox)
Debug.Print oFolder.Name
```

Очень часто требуется пройти циклом по всем папкам в почтовом ящике Outlook (а иногда даже в нескольких ящиках), просмотреть все сообщения в каждой папке (и во всех вложенных папках) и что-то сделать с этими сообщениями, например, собрать все письма от определенного отправителя, разбросанные по разным папкам, в одну специальную. В нашем примере мы просто пройдем циклом по всем папкам Outlook и выведем их имена, но, конечно, этот пример несложно переделать так, чтобы он выполнял какое-нибудь действие над ними в зависимости от условия. В нашем случае удобнее всего использовать две процедуры, одна из которых будет вызывать саму себя:

```
Public Sub StartProc1()
    Dim oOutlook As New Outlook.Application
    Dim oNamespace As Outlook.NameSpace
    Dim oChildFolder As Outlook.MAPIFolder
    Set oNamespace = oOutlook.GetNamespace("MAPI")

    'Перебираем каждую папку верхнего уровня и
    'вызываем для нее процедуру DoFolder()
    For Each oChildFolder In oNamespace.Folders
        DoFolder oChildFolder
    Next

End Sub

'Эта процедура выводит имя всех сложенных папок и
'для каждой из них опять вызывает саму себя
Public Sub DoFolder(ByVal oFolder As MAPIFolder)
    Dim oChildFolder As Outlook.MAPIFolder
    For Each oChildFolder In oFolder.Folders
        Debug.Print oChildFolder.Name
        'Для каждой вложенной папки опять вызываем процедуру DoFolder()
        DoFolder oChildFolder
    Next

End Sub
```

Далее представлена информация о свойствах и методах коллекции `Folders` и объекта `MAPIFolder`.

У коллекции `Folders` свойства и методы стандартные, как и у большинства коллекций (`Count`, `Item()`, `Add()`, `Remove()` и т. п.). Зато у объекта `MAPIFolder` важных свойств и методов очень много.

Самые важные свойства объекта `MAPIFolder` приведены в следующем списке.

- `AddressBookName` — позволяет поменять имя папки с контактами для отображения в адресной книге пользователя. Для других папок применяться не может (вернется ошибка).
- `CurrentView` — возвращает объект `View`, который определяет, как отображается данная папка для пользователя. Для объекта `MAPIFolder` это свойство доступно только для чтения.
- `DefaultItemType` — это свойство позволяет вернуть (в виде константного значения) тип элемента папки по умолчанию (почтовое сообщение, контакт и т. п.). Это свойство определяется при создании папки и после изменено быть не может. Обычно это свойство используется для исключения каких-то папок из обработки (чтобы, например, не искать почтовые сообщения в папке с контактами).
- `DefaultMessageClass` — то же самое, что и `DefaultItemType`, но информация возвращается не в виде числа, а в виде строкового значения. Выбирайте, что вам удобнее.
- `Description` — просто описание папки. При использовании графического интерфейса Outlook доступно через свойства папки.
- `EntryID` — это свойство очень удобно использовать как уникальный идентификатор сообщения, который создается автоматически при появлении сообщения в любом MAPI-совместимом хранилище (например, в файле Outlook или в почтовом ящике Exchange Server) и изменяется только при переносе в другое хранилище.
- `FolderPath` — полный путь к папке в иерархии хранилища Outlook, например, `"\\Личные папки\Входящие"`.
- `Folders` — очень важное свойство, которое возвращает коллекцию вложенных папок для данной папки. Как уже говорилось, очень часто используется для того, чтобы пройти циклом по всему дереву папок.
- `InAppFolderSyncObject` — определяет, будет ли эта папка синхронизироваться при работе специальной группы синхронизации (объекта `SyncObject`) под названием **Application Folders**. Эта группа синхронизации (она видна через меню **Сервис | Отправить/Получить**) отличается тем, что только ее можно изменять программным образом.
- `IsSharePointFolder` — позволяет определить, находится ли эта папка с контактами или элементами календаря на Windows SharePoint Services

(для обеспечения коллективной работы). Обычно используется для проверок.

- `Items` — еще одно важнейшее свойство. Обеспечивает доступ к коллекции `Items` всех элементов данной папки. Подробнее про работу с элементами папок — в *разд. 13.6*.
- `Name` — это, конечно, имя папки.
- `ShowAsOutlookAB` — определяет, показывать ли содержимое папки с элементами типа **Контакты** в окне выбора адреса при создании почтового сообщения. По умолчанию для всех папок с **Контактами** это свойство установлено в `True`. Обычно используется только тогда, когда какая-то папка с контактами используется для служебных целей.
- `ShowItemCount` — определяет, что будет показываться в строке сообщений приложения Outlook для папки: ничего, общее количество всех сообщений или общее количество только непрочитанных сообщений.
- `StoreID` — MAPI-совместимый уникальный идентификатор хранилища, в котором находится данная папка. Выглядит как очень длинная строка (516 символов). Можно использовать для распознавания, например, нескольких почтовых ящиков Exchange.
- `UnReadIemCount` — количество непрочитанных сообщений для данной папки. Доступно только для чтения.
- `Views` — возвращает коллекцию объектов `View` (режимов отображения), которые ассоциированы с данной папкой.
- `WebViewOn` — позволяет включить для папки отображение в виде HTML-страницы (которая может быть совершенно посторонней и никак не связанной с элементами Outlook и с самой этой папкой). Страница, которую нужно отобразить, задается при помощи свойства `WebViewURL`. Эту возможность можно включить и в графическом интерфейсе на вкладке **Домашняя страница** свойств папки.

То, что делают методы `CopyTo()`, `MoveTo()`, `Delete()`, `AddToFavorites()`, `Display()`, — понятно из их названий. Метод `GetExplorer()` позволяет вернуть объект `Explorer`, представляющий эту папку в Проводнике Outlook.

13.6. Коллекция *Items* и объекты элементов Outlook

Работа с элементами папок (почтовыми сообщениями, контактами, элементами календаря и т. п.) — это обычно самая важная часть программ, исполь-

зующих объектную модель Outlook. Именно с ними и приходится выполнять различные операции.

Доступ к элементам папок чаще всего производится через свойство `Items` объекта `MAPIFolder`, которое возвращает коллекцию `Items`. В этой коллекции находятся все элементы данной папки. Однако единого объекта для них не предусмотрено. Вместо этого в вашем распоряжении 16 отдельных объектов для каждого вида элементов в Outlook.

- `AppointmentItem` — то, что на графическом интерфейсе русского Outlook называется **Встречей**. Этот элемент обычно находится в папке **Календарь**.
- `ContactItem` — это контакт. Создавать новые контакты программным образом приходится очень часто.
- `DistList` — это еще один элемент, который обычно находится в папке **Контакты**. Он представляет собой список рассылки.
- `DocumentItem` — это любой файл, который помещен внутрь хранилища Outlook и не совпадает по своему формату ни с одним другим элементом Outlook. Объектом `DocumentItem` может быть, например, документ Word, книга Excel, ZIP-архив, файл Acrobat Reader PDF, исполняемый EXE-файл и т. п. — в общем, любой файл операционной системы. Однако увлекаться хранением файлов в хранилищах Outlook (файлах PST, а также в почтовых ящиках и общих папках Exchange Server), конечно, не стоит. Это замедлит доступ к обычным элементам Outlook. Такая возможность изначально была предусмотрена для того, чтобы, например, в общей папке Outlook для проекта вместе с перепиской по нему хранить также и различные, относящиеся к нему файлы.
- `JournalItem` — это запись в дневнике.
- `MailItem` — наиболее привычный многим элемент. Представляет собой сообщение электронной почты.
- `MeetingItem` — приглашение на встречу (специальный тип электронного сообщения). Обычно встречается там же, где и обычные сообщения электронной почты, например, в папке **Inbox**. Создать программным образом этот элемент невозможно — он создается только автоматически при получении соответствующего сообщения (отправить его можно при помощи объекта `AppointmentItem`, что соответствует элементу **Встреча** в **Календаре**).
- `NoteItem` — объект заметки (из папки **Заметки**). От всех других элементов отличается минимальным количеством свойств и методов.

- `PostItem` — еще одна специальная разновидность почтового сообщения. Это сообщение, которое отправлено в общую папку. От обычного объекта `MailItem` отличается тем, что:
  - встречается только в общих папках;
  - для его отправки вместо метода `Send()` используется метод `Post()`.
- `RemoteItem` — тоже очень специальная разновидность почтового сообщения. Этот объект представляет собой почтовое сообщение с минимальным количеством заполненных свойств (заполнена может быть только информация о получателе, отправителе, дате получения и размере сообщения) и текстом сообщения, в котором находятся первые 256 символов письма. Эти объекты создаются автоматически в тех ситуациях, когда Outlook подключается к почтовому ящику на сервере Exchange Server через соединение удаленного доступа (обычно коммутируемое). Если сообщение находится в файле OST (т. е. скачано с того же сервера Exchange Server по MAPI или получено по протоколу POP3/IMAP4), то этот объект никогда для него не создается.
- `ReportItem` — специальное почтовое сообщение, представляющее собой специально сгенерированное служебное письмо: обычно это сообщение о невозможности доставки *(non-delivery report)*, созданное вашим почтовым сервером, о задержке в передаче сообщения или о другой ошибке. Эти объекты также нельзя создавать программным образом, они создаются только автоматически при получении сообщения такого типа.
- `TaskItem` — это задача или поручение из папки **Задачи**.
- `TaskRequestAcceptItem`, `TaskRequestDeclineItem`, `TaskRequestItem`, `TaskRequestUpdateItem` — это специальные почтовые сообщения, которые относятся к переписке по поводу делегирования задач. Эти объекты также нельзя создавать программным образом.

Надо сказать, что в подавляющем большинстве случаев вас будут интересовать только объекты `MailItem`, `ContactItem` и иногда `DocumentItem`. На их рассмотрении мы и сосредоточимся.

И еще один очень важный момент. Как уже говорилось ранее, в объектную модель Outlook встроены специальные ограничения, которые призваны не дать вирусам использовать возможности этой объектной модели в своих интересах. Как правило, это самые важные свойства, в которых содержится информация об адресах электронной почты, имени отправителя, тексте писем и т. п. Эти ограничения встроены в следующие объекты: `AppointmentItem`, `ContactItem`, `MailItem` и `TaskItem`, т. е. во все основные объекты элементов Outlook. Ограничения наложены и на некоторые действия, которые могут выполняться с этими объектами, например, на отправку писем. Так, создание

и отправка электронного сообщения средствами Outlook выглядит очень просто:

```
Dim oOutlook As New Outlook.Application
Dim oMessage As Outlook.MailItem
'Создаем объект сообщения
Set oMessage = oOutlook.CreateItem(olMailItem)
'Кому
oMessage.To = "Administrator@nwtraders.msft"
'Тема сообщения
oMessage.Subject = "Привет из VBA"
'Текст сообщения. Использование свойства Body означает,
'что мы посылаем сообщение обычным текстом.
'Можно также послать сообщение в формате HTML или RTF
oMessage.Body = "Текст сообщения"
'Добавляем вложение
oMessage.Attachments.Add ("C:\installlog.txt")
'Отправляем сообщение
oMessage.Send
```

Однако чтобы усложнить жизнь вирусам, вместо простой отправки сообщения появляется окно, аналогичное представленному на рис. 13.3. Более того, чтобы добиться окончательной победы над вирусами, кнопка **Да** на протяжении нескольких секунд будет недоступна.

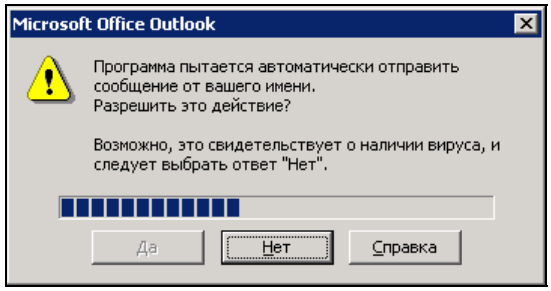


Рис. 13.3. Предупреждающее окно в Outlook

Что же делать в такой ситуации?

Варианты могут быть разными:

- первый вариант — *отправлять сообщение из макроса, который находится в модуле VBA самого Outlook* (без программного открытия Outlook). В этом случае такое окно появляться не будет. Но такое решение обычно не очень удобно, особенно с точки зрения переноса этого программного кода для запуска на разных компьютерах;

- второй вариант — *использовать специальные заменители объектов Outlook*, которые не выдают таких сообщений и позволяют разработчикам отправлять сообщения из VBA без проблем. Самое распространенное и рекомендованное средство такого рода — Outlook Redemption. Получить информацию о нем и бесплатно скачать его можно с сайта **http:// www.dimastr.com/redemption**. Этот вариант вполне удобен, если вы занимаетесь рассылкой электронной почты с одного сервера, но устанавливать его на все компьютеры, конечно, хлопотно;
- третий вариант — *программным образом обнаруживать окна подтверждений и также программно нажимать в них нужные кнопки*. Однако разработчики Microsoft позаботились о том, чтобы из кода VBA или VBScript сделать это было невозможно. В принципе, такие операции вполне можно выполнить при помощи низкоуровнего программного интерфейса Windows API (или, как вариант, получить доступ к этим возможностям при помощи специальных средств специального программного интерфейса для работы с сообщениями MAPI). Однако вам придется использоваться вместо VBA другими языками программирования (C++ или Delphi), поскольку VBA не поддерживает все нужные типы данных для работы с API. Кроме того, для уверенной работы с API требуются специальные знания в области системного программирования. Дополнительную информацию об использовании таких приемов при работе с Outlook можно получить из статьи **http://www.mapilab.com/ru/support/articles /vb_outlook_security_1.html** (на русском языке);
- четвертый вариант (с моей точки зрения наиболее удобный) — *использовать для рассылки и программной обработки входящих писем объектную библиотеку CDO*, которая есть на любом компьютере под управлением Windows 2000, XP и 2003. Подробнее о применении этой библиотеки будет рассказано в *разд. 13.8*.

Теперь о самих объектах элементов Outlook. Свойств и методов у этих объектов очень много (например, у объекта `ContactItem` почти 100 свойств), и на рассмотрение всех их потребовалось бы очень много времени. Тем не менее большинство свойств этих объектов очевидны и проблем при их использовании возникнуть не должно. Обычно единственный вопрос, который может возникнуть, — какое программное свойство соответствует определенному атрибуту элемента. К сожалению, макрорекордера в Outlook не предусмотрено, но понять, как называется то или иное свойство, можно при помощи окна **Locals**. Его использование рассмотрим на примере. Пусть вам нужно найти, какому программному свойству объекта `ContactItem` соответствует поле **Краткое имя**.

Первое, что нам нужно сделать, — это создать контакт, в котором было бы заполнено данное свойство, например, строкой `"Краткое имя"` (рис. 13.4).

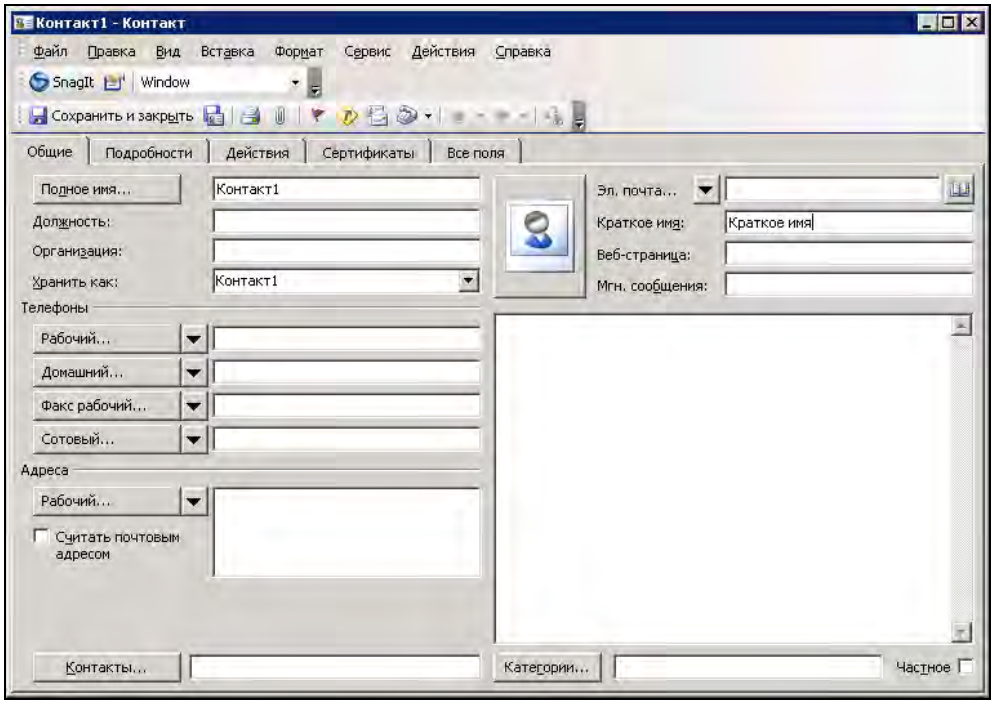


Рис. 13.4. Создаем контакт

Следующее, что нужно сделать, — написать программный код, в котором был бы создан объект данного элемента. Например, если этот контакт находится в папке **Контакты**, то код (из макроса в Outlook) может быть таким:

```
Dim oNamespace As NameSpace
Dim oFolder As MAPIFolder
Dim oContactItem As ContactItem

Set oNamespace = Application.GetNamespace("MAPI")
Set oFolder = oNamespace.GetDefaultFolder(olFolderContacts)
Set oContactItem = oFolder.Items("Контакт1")
Stop
```

Команда `Stop` здесь предназначена для того, чтобы остановиться в нужном месте.

Затем нужно запустить код на выполнение и, когда он остановится на последней строке, открыть (при помощи меню **View**) окно **Locals**.

В нашем случае вы увидите в нем четыре объекта (рис. 13.5).

Конечно же, нам нужно развернуть узел объекта `oContactItem` и поискать его свойство, для которого установлено значение `"Краткое имя"`. К своему

удивлению, мы можем обнаружить, что это свойство называется `EMail1DisplayName` (рис. 13.6).

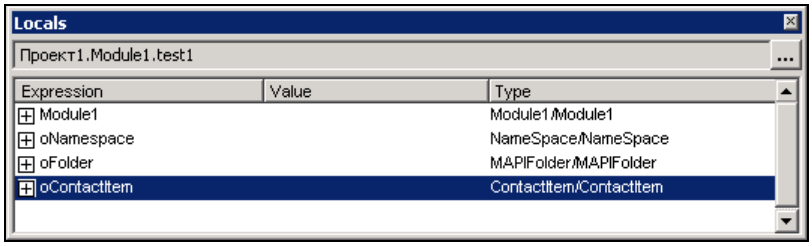


Рис. 13.5. Окно **Locals** с объектом контакта

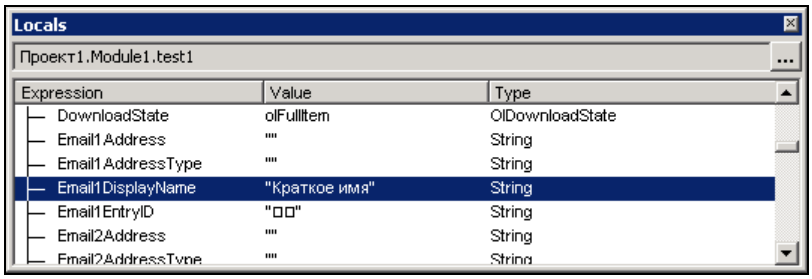


Рис. 13.6. Свойство с заполненным нами значением в окне **Locals**

Таким же образом можно находить нужные свойства и для других объектов `Item`.

13.7. Другие объекты Outlook

Мы прошли с вами только по главной ветви объектной модели Outlook: `Application` — `Namespace` — `Folders/MAPIFolder` — *`объекты_элементов`*. Но за ее пределами находится еще очень много удобных в применении объектов, о которых будет рассказано в этом разделе.

Объект `Explorer` предназначен для отображения папки в интерфейсе Outlook. При этом вы можете выбрать для отображения встроенное или пользовательское представление (при помощи объекта `View`). Например, чтобы программным образом открыть папку **Контакты** и рассортировать в ней объекты контактов по организациям, можно использовать следующий код:

```
Dim oNamespace As NameSpace
Dim oFolder As MAPIFolder
Dim oExplorer As Explorer
```

```
Set oNamespace = Application.GetNamespace("MAPI")
Set oFolder = oNamespace.GetDefaultFolder(olFolderContacts)
Set oExplorer = oFolder.GetExplorer()
oExplorer.Display
oExplorer.CurrentView = "По организациям"
```

Просмотреть все представления, доступные для данной папки (для них используется объект `View`), можно так:

```
Dim oNamespace As NameSpace
Dim oFolder As MAPIFolder
Dim oView As View

Set oNamespace = Application.GetNamespace("MAPI")
Set oFolder = oNamespace.GetDefaultFolder(olFolderContacts)
For Each oView In oFolder.Views
    Debug.Print oView.Name
Next
```

Еще одна полезная возможность объекта `Explorer` — определение того, какие элементы выделил пользователь. Например, чтобы получить информацию о темах всех писем, которые в настоящее время выделил пользователь в текущем окне, можно использовать код:

```
For Each Item In Application.ActiveExplorer.Selection
    If TypeName(Item) = "MailItem" Then Debug.Print Item.Subject
Next
```

Брат-близнец объекта `Explorer` — объект `Inspector`. Он также представляет окно Outlook, но уже с открытым на просмотр или редактирование элементом (почтовым сообщением, контактом и т. п.). Получение на него ссылки выглядит точно так же, как и для объекта `Explorer`, но используется он реже, в основном для проверок, не открыт ли обрабатываемый программно элемент пользователем в Outlook.

Отдельную ветвь представляют объекты для поиска в Outlook. Производить поиск в Outlook приходится очень часто. Можно реализовать свой собственный поиск простым перебором всех папок и элементов в них, начиная с верхнего уровня. Такой пример был приведен в *разд. 13.5*, посвященном работе с коллекцией `Folders` и объектами `MAPIFolder`. Более эффективная, но несколько более сложная возможность — это применение для поиска встроенных средств Outlook. Для этого используются два объекта — `Search` и `Results`, один метод `AdvancedSearch()` объекта `Application` и одно событие `AdvancedSearchComplete` (поскольку поиск работает в асинхронном режиме и

можно одновременно запускать до 100 поисков, как программно, так и из графического интерфейса). Выглядит это следующим образом.

Вначале запускаем на выполнение метод `AdvancedSearch()`. Этот метод принимает четыре параметра:

- `Scope` — диапазон поиска, т. е. имя папки. Если вы точно не знаете, где вам нужно искать, можно получить имя папки программно при помощи свойства `Folders` объектов `NameSpace` и `MAPIFolder`. При использовании специальных символов в имени папки рекомендуется заключать его в одинарные кавычки;
- `Filter` — самый сложный параметр. Определяет, что именно мы будем искать. Синтаксис должен соответствовать синтаксису фильтров при запросах на SQL Server (можно использовать в том числе и замечательный оператор `LIKE`), что, в принципе, могло быть очень удобным. Однако определить, как должны выглядеть имена столбцов, не так-то просто — скорее всего, за справкой придется обращаться на сайт Microsoft по адресам:
  - **http://msdn.microsoft.com/library/en-us/cdosys/html/_cdosys_schema_mailheader.asp**;
  - **http://msdn.microsoft.com/library/en-us/cdosys/html/_cdosys_schema_httpmail.asp**.

  Например, если мы производим поиск по теме сообщения (пытаясь найти все письма, в названиях которых встречается слово "Отчет"), то строка фильтра должна выглядеть так:

  ```
  "urn:schemas:mailheader:subject LIKE '%Отчет%'"
  ```

  Если же ищем точную тему "Отчет", то так:

  ```
  "urn:schemas:mailheader:subject = 'Отчет'"
  ```

- `SearchSubFolders` — значение этого параметра нужно установить в `True`, если вам нужно пройти и по всем вложенным папкам.
- `Tag` — просто строковый идентификатор поиска. Используется только тогда, когда вы одновременно запускаете несколько поисков, и вам нужно отличить результаты одного от результата другого.

В итоге метод `AdvancedSearch()` вернет нам объект `Search`:

```
Dim oSearch As Search
Set oSearch = Application.AdvancedSearch("Inbox", _
    "urn:schemas:mailheader:subject LIKE '%Отчет%'", True, "Search1")
```

Поскольку поиск выполняется в асинхронном режиме, процедура, из которой мы вызвали метод `AdvancedSearch()`, продолжит выполняться дальше, не до-

жидаясь его завершения. А нам придется отслеживать окончание поиска при помощи события `AdvancedSearchComplete`. Для этого в **Project Explorer** раскройте контейнер **Microsoft Office Outlook Objects**, щелкните два раза левой кнопкой мыши по строке **ThisOutlookSession** и в списке объектов и событий в верхней части окна редактора кода выберите `Application` и `AdvancedSearchComplete()` (рис. 13.7).

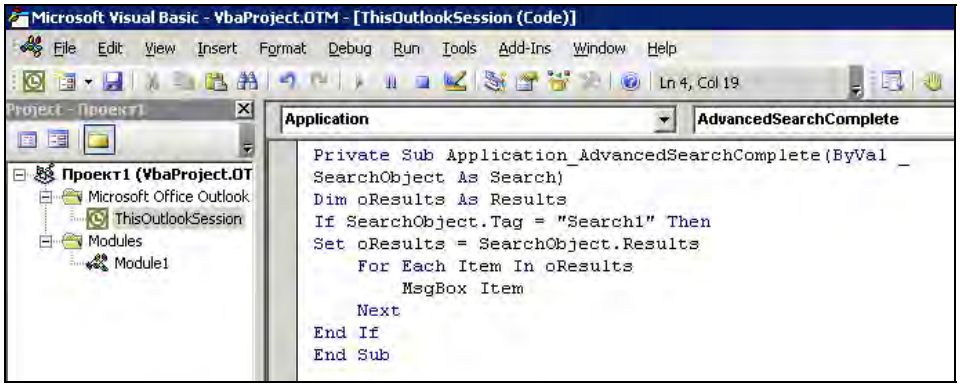


Рис. 13.7. Используем событие `AdvancedSearchComplete`

А в этом событии используем объект `Search` и производный от него объект `Results` (код показан на этом же рисунке). `Item` — это обычные элементы Outlook (в данном случае объекты `MailItem`), и вам вполне доступны все их свойства и методы.

13.8. Альтернатива при работе с электронной почтой — объектная библиотека CDO

Все-таки Outlook — это, прежде всего, программа для работы с электронной почтой, и если вы обратились к ее объектной модели, то для автоматизации именно операций с электронной почтой. Как уже было сказано ранее, нормальной работе с электронной почтой сильно мешают ограничения безопасности, встроенные в объектную модель Outlook. С ними можно бороться (как описано в *разд. 13.6*), а можно просто обойти, используя для отправки электронной почты специальную объектную модель CDO, в которой этих ограничений нет. Эту объектную модель можно использовать в том числе и из Outlook.

CDO (*Collaboration Data Objects*, объекты для совместной работы с данными) — это специальный набор библиотек для работы с электронной почтой и для администрирования сервера Exchange Server. Существует множество вер-

сий и разновидностей библиотек, которые входят в набор CDO, но нас интересует только одна: Microsoft CDO for Exchange 2000 Library, которая устанавливается вместе с Microsoft Office. Первое, что нужно будет сделать, — это добавить ссылку на эту библиотеку при помощи меню **Tools | References** в редакторе VBA.

Самый простой вариант отправки почты средствами CDO выглядит так:

```
Dim oMyMail As New CDO.Message
oMyMail.To = "Administrator@nwtraders.msft"
oMyMail.From = "Administrator@nwtraders.msft"
oMyMail.Subject = "Hello from CDO"
oMyMail.TextBody = "Our letter"
oMyMail.AddAttachment "C:\1.txt"
oMyMail.Send
```

Однако этот вариант с параметрами по умолчанию будет работать только в том случае, если на вашем компьютере установлены Exchange Server 2000/2003 или Internet Information Server с настроенной службой SMTP, поскольку физически сообщение просто будет помещено в каталог C:\Inetpub\mailroot\Pickup (по умолчанию), откуда его должна забрать служба Exchange Server или IIS. Однако есть и более удобный способ отправки сообщений через любой почтовый сервер, который поддерживает протокол SMTP. Для этого перед вызовом метода `Send()` мы должны настроить параметры отправки:

```
oMyMail.Configuration.Fields.Item _
  ("http://schemas.microsoft.com/cdo/configuration/sendusing") = 2
```

Здесь `"http://schemas.microsoft.com/cdo/configuration/sendusing"` — это название поля, и оно должно быть указано точно. По умолчанию значение этого поля равно 1, что означает использование каталога Pickup.

Указать почтовый сервер можно так:

```
omyMail.Configuration.Fields.Item _
  ("http://schemas.microsoft.com/cdo/configuration/smtpserver") = _
   "smtp.YourServer.com"
```

Настройка режима аутентификации производится при помощи того же объекта `CDO.Configuration`:

```
oMyMail.Configuration.Fields.Item _
("http://schemas.microsoft.com/cdo/configuration/smtpauthenticate") = 1
```

Значение 1 означает, что используется базовая аутентификация, значение 0 — без аутентификации (анонимно), значение 2 — аутентификация NTLM.

Имя пользователя и пароль можно передать точно так же:

```
oMyMail.Configuration.Fields.Item _
  ("http://schemas.microsoft.com/cdo/configuration/sendusername") = _
  "YourLogin@YourDomain.com"
oMyMail.Configuration.Fields.Item _
  ("http://schemas.microsoft.com/cdo/configuration/sendpassword") = _
  "Password"
```

Иногда необходимо также определить использование специфического порта (отличного от 25), будет или нет использоваться SSL и время тайм-аута:

```
oMyMail.Configuration.Fields.Item _
("http://schemas.microsoft.com/cdo/configuration/smtpserverport") = 2525

oMyMail.Configuration.Fields.Item _
("http://schemas.microsoft.com/cdo/configuration/smtpusessl") = False

oMyMail.Configuration.Fields.Item _
("http://schemas.microsoft.com/cdo/configuration/smtpconnectiontimeout") =
60
```

После любых изменений, вносимых в конфигурацию, их вначале надо сохранить:

```
oMyMail.Configuration.Fields.Update
```

и только после вызывать метод `Send()`:

```
oMyMail.Send
```

Если возникает проблема с кодировками (обычно, если в системе установлен русский язык, не возникает), но можно еще перед отправкой добавить строку вида:

```
oMyMail.TextBodyPart.Charset = "windows-1251"
```

К сожалению, эта библиотека работает только с протоколом SMTP и каталогом Pickup на диске. Она не умеет работать ни с протоколом POP3, ни с IMAP4, ни с MAPI, а значит, подключиться к почтовому серверу и проверить на нем появление новых сообщений (как в нашем примере с Outlook) мы не сможем. Придется использовать другую библиотеку из набора CDO — Microsoft CDO 1.21 Library. Она умеет работать только с MAPI (т. е. с Exchange Server), зато может выполнять различные операции в почтовом ящике на Exchange Server без всяких предупреждающих сообщений.

Вначале, как всегда, нужно добавить ссылку на библиотеку Microsoft CDO 1.21 Library (меню **Tools | References**). Чтобы, например, отследить появление новых писем, нужно выполнить следующий код:

```
Dim oSession As New MAPI.Session
Dim oFolder As MAPI.Folder
Dim oMessage As MAPI.Message

'"Outlook" — имя почтового профиля. Если этот параметр не передать,
'то возникнет диалоговое окно с предложением выбрать нужный профиль.
'Имя нужного профиля можно узнать как раз из этого диалогового окна
oSession.Logon ("Outlook")
Set oFolder = oSession.Inbox
For Each oMessage In oFolder.Messages
    If oMessage.Unread = True Then Debug.Print oMessage
Next
```

Полную справку по этим объектным моделям можно прочитать в MSDN.

Задание для самостоятельной работы 13: Работа с контактами в Outlook

Ситуация:

В приложении для работы с клиентами вашего предприятия есть таблица со списком организаций, в которой хранится информация о представителях этих организаций и их координатах. Сотрудники вашего предприятия, которые осуществляют связь с клиентами, просят сделать так, чтобы эта информация стала доступной для них из Outlook.

ЗАДАНИЕ:

Напишите макрос Outlook `ImportContacts()`, который бы:

1. Создавал новую папку Outlook с элементами типа **Контакт** под именем **Контакты клиентов**.
2. Создавал в этой папке контакты для всех записей таблицы `Клиенты` базы данных C:\Program Files\Microsoft Office\OFFICE11\SAMPLES\Борей.mdb.

   Чтобы упростить задачу, достаточно поместить в создаваемые контакты только информацию о названии организации, имени представителя и телефоне.

Примечание

На практике, вполне возможно, удобнее было бы создать общую папку на сервере Exchange Server и программным образом создать в ней контакты, чтобы можно было предоставить доступ к ним сразу группе пользователей. В нашем случае для простоты мы создаем элементы контактов в локальной папке Outlook. Код VBA для создания контактов в любом случае будет идентичен.

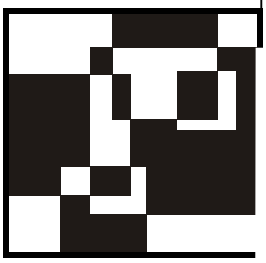
Ответ к заданию 13

1. Запустите Outlook и нажмите в нем клавиши <Alt>+<F11>, чтобы открыть редактор кода Visual Basic.
2. В меню **Tools | References** добавьте ссылку на объектную библиотеку Microsoft ActiveX Data Objects 2.1 Library.
3. В окне **Project Explorer** щелкните правой кнопкой мыши по объекту проекта **Проект1** и в контекстном меню выберите **Insert | Module**. Будет создан новый стандартный модуль **Module1**.
4. В этом модуле создайте новую процедуру `ImportContacts()` и добавьте в нее необходимый код. Он может быть таким:

```
Public Sub ImportContacts()
    Dim oFolder As MAPIFolder
    Dim oNameSpace As NameSpace
    Dim oContact As ContactItem

    Set oNameSpace = Application.GetNamespace("MAPI")
    'Создаем папку и получаем ссылку на ее объект
    Set oFolder = oNameSpace.Folders("Личные папки").Folders. _
        Add("Контакты клиентов", olFolderContacts)

    'Создаем объект соединения
    Dim cn As New ADODB.Connection
    cn.ConnectionString = "Provider=Microsoft.Jet.OLEDB.4.0;" _
& "Data Source=C:\Program Files\Microsoft
Office\OFFICE11\SAMPLES\Борей.mdb"
    cn.Open

    'Создаем объект Recordset
    Dim rs As New ADODB.Recordset
    rs.Open "Клиенты", cn

    'Проходим циклом по Recordset
    Do While rs.EOF = False
        'Создаем объект контакта
        Set oContact = Application.CreateItem(olContactItem)
        'Заполняем его свойства на основе данных из Recordset
        oContact.CompanyName = rs.Fields("Название")
        oContact.FullName = rs.Fields("ОбращатьсяК")
        oContact.BusinessTelephoneNumber = rs.Fields("Телефон")
        'Перемещаем в нашу папку и сохраняем
        oContact.Move oFolder
        oContact.Save
```

```
            'Освобождаем память
            Set oContact = Nothing
            'Сдвигаемся на одну запись в Recordset
            rs.MoveNext
        Loop

    End Sub
```

ГЛАВА 14



Программирование в PowerPoint

PowerPoint — это программа для работы с презентациями (т. е. с наборами графических изображений — слайдов, иногда со звуковым сопровождением). Использовать возможности VBA в PowerPoint на предприятиях приходится намного реже, чем возможности Word или Excel, однако иногда такая необходимость возникает. Часто специалисты используют презентации PowerPoint для сопровождения выступлений при демонстрации продуктов или услуг, отчетов о деятельности и т. п. Поскольку со слайдами можно связывать звуковое сопровождение, PowerPoint активно используется для целей обучения, например, для подготовки интерактивных уроков. Еще одна часто используемая возможность — создание звуковых книг с картинками для детей. При помощи PowerPoint можно создавать фотоальбомы со звуковым сопровождением, диафильмы со звуком, детские игры и многое другое. И как только данных становится много (например, цифровых фотографий может быть несколько сотен) сразу встает вопрос об автоматизации.

Чаще всего приходится программным способом выполнять следующие действия в PowerPoint:

- *автоматически создавать презентации* (например, на основе набора изображений в каталоге);
- *производить обработку презентаций* — менять формат изображений, добавлять или изменять аудиосопровождение и т. п. Чаще всего подобные действия приходится производить в тех ситуациях, когда презентации были связаны с внешними файлами и эти файлы изменились.

В PowerPoint система объектов выглядит следующим образом:

- объект самого высокого уровня — `Application` с набором свойств и методов, очень похожим на аналогичные объекты в Word и Excel;

- уровень ниже — коллекция `Presentations` с объектами `Presentation`. Можно сказать, что эти объекты по месту в иерархии примерно аналогичны объекту `Workbook` в Excel;
- в объект `Presentation` встроена коллекция `Slides` с объектами `Slide`. В качестве аналога можно привести листы `Worksheet` в книгах Excel;
- в объект `Slide` встроена коллекция `Shapes` с объектами `Shape`. Объекты `Shape` представляют собой все элементы слайда (всего их 22 типа: изображение, надпись, диаграмма, заголовок, таблица, автофигура и т. п.).

Вокруг этих четырех объектов — `Application`, `Presentation`, `Slide` и `Shape` — и строится вся объектная модель PowerPoint.

В этой главе мы не будем приводить справку по свойствам и методам различных объектов PowerPoint (их можно быстро найти при помощи макрорекордера), а вместо этого проиллюстрируем работу с PowerPoint на примерах из практики.

Предположим, что нам нужно создать презентацию PowerPoint на основе набора JPG-картинок, которые будут лежать в каталоге C:\Slides (например, они получены со сканера или цифрового фотоаппарата). Имена JPG-файлов следуют по порядку, например, с DSCN2440.JPG по DSCN2480.JPG. Файлов в каталоге может быть произвольное количество, поэтому нам нужно взять все файлы из этого каталога. Наша задача — поместить их в презентацию по порядку. Задача усугубляется тем, что JPG-файлы разного размера (по высоте и ширине), а слайды желательно сделать одинаковыми.

Как ни удивительно, код VBA для PowerPoint удобнее запускать не из PowerPoint, а из внешнего приложения, поддерживающего VBA, например, из Word или Excel. Таким образом, на момент запуска у нас гарантированно не будет активных презентаций и мы ничего не перепутаем при вставке.

Наше решение может выглядеть следующим образом:

1. Создаем новый документ в Word или Excel, в него помещаем кнопку или обеспечиваем другой графический интерфейс по вкусу. Главное — это не забыть добавить в проект ссылки на две объектные библиотеки:
   - *Microsoft PowerPoint 11.0 Object Library* (msppt.olb) — для объектов самого PowerPoint;
   - *Microsoft Scripting Runtime* (ScrRun.dll) — для того, чтобы можно было пользоваться объектом `FileSystemObject` и другими возможностями для работы с файловой системой. Эта библиотека, которая есть на любом компьютере начиная с Windows 2000, — самый удобный способ выполнения большинства действий в файловой системе.

   Далее можно приступать к созданию кода.

2. Конечно, первое, что нам потребуется — запустить PowerPoint. Делается это точно так же, как и для Word, Excel, Access и т. п.:

```
Dim oApp As New PowerPoint.Application
oApp.Activate
oApp.Visible = msoTrue
```

3. Следующее действие — создание новой пустой презентации:

```
Dim oPresent As PowerPoint.Presentation
Set oPresent = oApp.Presentations.Add()
```

   Все абсолютно стандартно, как будто мы создаем новый документ Word. А вот дальше начинаются моменты, специфические для PowerPoint.

4. Следующим действием должно быть создание слайда. Но нам придется создавать столько слайдов, сколько файлов находится в каталоге C:\Slides. Конечно, слайды будут создаваться в цикле. Вначале мы получаем при помощи библиотеки Scripting Runtime (можно было бы обойтись и средствами Office, но так проще) коллекцию всех файлов этого каталога:

```
Dim oFSO As New Scripting.FileSystemObject
Dim oFolder As Scripting.Folder
Dim oFile As Scripting.File

Set oFolder = oFSO.GetFolder("C:\Slides")
For Each oFile In oFolder.Files
    ...
Next
```

   Если мы вместо многоточия вставим строку, например, такого вида:

```
MsgBox oFile.Name
```

   то можно будет убедиться, что мы получили набор файлов в правильном порядке.

5. Далее нам все-таки нужно создать слайды. Делается это при помощи метода `Add()` коллекции `Slides`. В документации к русскому PowerPoint 2003 описание этого метода по непонятной причине отсутствует (несмотря на то, что справка по VBA все равно приводится на английском), но из всплывающей подсказки можно догадаться, что этот метод принимает два обязательных параметра: номер слайда в презентации (нумерация должна начинаться с 1), и одно из значений перечисления `ppSlideLayout` (из нескольких десятков), которое определяет шаблон слайда.

   Номер слайда придется обеспечивать счетчиком, а лучший для нас шаблон — пустой:

```
Dim nCounter As Integer
nCounter = 1
For Each oFile In oFolder.Files
    Set oSlide = oApp.ActivePresentation.Slides.Add(nCounter, _
        ppLayoutBlank)
    ...
    nCounter = nCounter + 1
Next
```

6. А теперь самое главное — вставляем в слайд изображение и настраиваем его размеры. Для этой цели можно использовать метод `AddPicture()` коллекции `Shapes` каждого слайда:

```
oSlide.Shapes.AddPicture FileName:="C:\Slides\" & oFile.Name, _
    LinkToFile:=msoFalse, SaveWithDocument:=msoTrue, _
    Left:=10, Top:=10, Width:=700, Height:=520
```

   Параметр `FileName` — это имя передаваемого файла. Именно он и будет меняться в цикле. Параметр `LinkToFile` определяет, будет ли файл изображения помещен внутрь презентации (`msoFalse`) или в презентацию будет помещена ссылка на него (`msoTrue`). Конечно, если вставляемые файлы не очень большие, то с точки зрения удобства и производительности презентации лучше помещать изображения внутрь презентации (файла PPT). Параметр `SaveWithDocument` определяет, сохранять ли изображения вместе с презентацией (в нашем случае сохранять). А параметры `Left`, `Top`, `Width` и `Height` нужны, чтобы сделать изображения одинакового размера (нужные значения подбираются вручную).

   Естественно, код этого пункта помещается вместо многоточия в цикл пункта 5. Чтобы не возиться с удалением обработанных файлов, я бы поместил в цикл еще одну очевидную строку:

```
oFile.Delete
```

Итоговый код для нашей задачи может выглядеть так:

```
Dim oApp As New PowerPoint.Application
oApp.Activate
oApp.Visible = msoTrue

Dim oPresent As PowerPoint.Presentation
Set oPresent = oApp.Presentations.Add()

Dim oFSO As New Scripting.FileSystemObject
Dim oFolder As Scripting.Folder
Dim oFile As Scripting.File
```

```
Set oFolder = oFSO.GetFolder("C:\Slides")
For Each oFile In oFolder.Files
    Set oSlide = oApp.ActivePresentation.Slides.Add(nCounter, _
        ppLayoutBlank)
    oSlide.Shapes.AddPicture FileName:="C:\Slides\" & oFile.Name, _
        LinkToFile:=msoFalse, SaveWithDocument:=msoTrue, _
        Left:=10, Top:=10, Width:=700, Height:=520
    oFile.Delete
Next
```

Несколько строк кода могут заменить часы нудной работы по копированию и вставке изображений вручную.

На практике код для создания эффектов анимации, звукового сопровождения, диапазонов фигур и т. п. может оказаться очень сложным. Найти в документации то, что вам нужно, не так-то просто. Рекомендуется для получения "наводящих указаний" активно использовать макрорекордер и анализировать созданный им код. Однако макрорекордер часто выбирает какие-то очень нетривиальные способы выполнения различных действий. Например, для вставки того же рисунка он предлагает использовать код типа:

```
ActiveWindow.Selection.SlideRange.Shapes.AddPicture ...
```

что, конечно, задачу не упрощает. Так что код макрорекордера всегда рекомендуется проверять и исправлять.

Задание для самостоятельной работы 14: Программное добавление элементов в слайды

ЗАДАНИЕ:

Создайте макрос в PowerPoint, который бы добавлял во все слайды активной презентации в правый нижний угол надпись "© Академия специальных курсов, 2005" (рис. 14.1).

Примечание

В реальной работе, возможно, удобнее было поместить этот макрос по внешнее приложение VBA, например, в документ Word или лист Excel, чтобы не копировать этот код для каждой новой презентации. Но в этой работе для простоты код будет выполняться из самого PowerPoint.

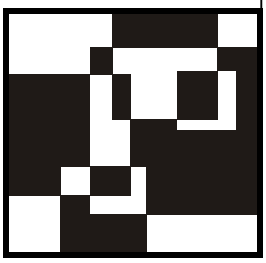
Рис. 14.1. Часть слайда с копирайтом

Ответ к заданию 14

Код макроса может быть, например, таким:

```
Public Sub InsertCopyRight()
    Dim oSlide As Slide
    Dim oShape As Shape

    'Проходим циклом по всем слайдам в презентации
    For Each oSlide In Application.ActivePresentation.Slides

        'Для каждого слайда создаем надпись (в виде объекта Shape)
        'Нужные значения для числовых параметров Left, Top, Width и
        'Height находим подбором или через макрорекордер
        Set oShape = oSlide.Shapes.AddTextbox(_
            msoTextOrientationHorizontal, 500, 510, 210, 40)

        'Доступ к тексту через вложенные объекты TextFrame и TextRange
        oShape.TextFrame.TextRange.Text = Chr(169) & _
            " Академия специальных курсов, 2005"

        'Занимаемся украшательством
        oShape.TextFrame.TextRange.Font.Size = 12
        oShape.TextFrame.TextRange.Font.Bold = msoTrue
    Next

End Sub
```

ГЛАВА 15



Программирование в Project

15.1. Основы программирования в Project Professional. Объект *Application*

Еще одно приложение в семействе Office — это Project. Он часто используется на больших предприятиях. Как правило, если на предприятии применяется Project (чаще всего вместе с Project Server), то значит, данных много и, скорее всего, для выполнения различных операций потребуется автоматизация.

Самый главный объект в Project Professional 2003, как и во всех других приложениях Office, — это объект `Application` со стандартным набором свойств и методов. Создать объект `Application` — значит запустить Project Professional:

```
Dim pj As New MSProject.Application
pj.Visible = True
MsgBox pj.Name & " " & pj.Version
```

Если запуск производится из другого приложения Office, то вам потребуется добавить в проект ссылку на библиотеку Microsoft Project 11.0 Object Library (меню **Tools | References**).

Если мы выполняем код из самого Project, то объект `Application` будет доступен по умолчанию. Если мы не укажем, из какого именно объекта используется свойство или метод, то компилятор VBA будет считать, что это свойство или метод объекта `Application`. Например, при вызове из VBA в Project следующие строки кода идентичны:

```
Application.ActiveProject.Visible = True
```

и

```
ActiveProject.Visible = True
```

В Project Professional есть замечательное средство, которого очень не хватает в Word, Excel, PowerPoint и в других приложениях Office. Это средство — глобальный корпоративный шаблон. Он единый для всего сервера Project Server, автоматически загружается при подключении пользователя к серверу и модули из него доступны всем пользователям Project Server. Фактически это корпоративное хранилище кода VBA. Конечно, этот шаблон можно использовать не только для действий, непосредственно связанных с Project, но и для других целей, например, для создания документов Word по единой форме. Немного развив эту тему, можно прийти к системе корпоративного репортинга с централизованным хранилищем шаблонов отчетов.

Однако при этом нужно быть очень внимательным: если у вас есть модуль **Module1** в глобальном корпоративном шаблоне и модуль с таким же именем в проекте, то модуль проекта со всем кодом будет заменен модулем из шаблона.

15.2. Коллекция *Projects*, объект *Project* и вложенные объекты

После того, как объект `Application` создан (а значит, открыт Project, программно или вручную), нам нужно создать или открыть в нем проект.

Создание проекта может выглядеть очень просто, при помощи метода `Add()` коллекции `Projects`:

```
Dim oProj As Project
Set oProj = Projects.Add()
```

Дополнительные необязательные параметры метода `Add()` позволяют создать новый проект на основе шаблона или открыть окно для выбора шаблонов.

Если нужно открыть существующий проект, то нам может встретиться два варианта этой задачи:

1. Нужно открыть локальный проект (из файла mpp).

   Здесь все просто:

   ```
   Application.FileOpen "D:\Project1.mpp"
   ```

   А затем ловим объектную ссылку на этот проект (поскольку сам метод `FileOpen()` ее не возвращает):

   ```
   For Each Project In Application.Projects
       If Project.Name = "Project1.mpp" Then
           Set oProj = Project
           Exit For
       End If
   ```

```
Next
MsgBox oProj.Name
```

2. Нужно открыть проект с сервера Project Server. Здесь все выполняется точно так же, за исключением того, что мы используем метод `FileOpen()` с параметром с хитрым синтаксисом:

   ```
   Application.FileOpen "<>\Забор.Опубликовано"
   ```

   `"<>\Забор.Опубликовано"` — это, конечно, имя проекта на сервере Project Server. К этому серверу мы должны подключиться в момент запуска Project. При этом при запуске Project просто указать сервер, к которому мы подключаемся, не получится. Придется работать с коллекцией `Profiles` объекта `Application`, который позволяет просмотреть профили, создать новый профиль, настроить параметры подключения (путь к серверу, имя и тип учетной записи), сделать его профилем по умолчанию и т. п. Например, если мы работаем из внешнего приложения, можно создать нужный профиль, сделать его профилем по умолчанию, подключиться к Project Server, а потом вернуть все обратно. Но обычно при использовании Project Server на предприятии пользователь автоматически подключается к нему при каждом запуске Project Professional (в том числе и программном), поэтому выбирать профиль нам не нужно.

   Такой же синтаксис (`"<>\Забор.Опубликовано"`) можно будет использовать и потом, при сохранении проекта на Project Server (методом `SaveAs()` объекта `Project`).

После того как мы создали проект, можно работать с его элементами: задачами, ресурсами и назначениями.

Добавление задачи в проект может выглядеть так:

```
Dim oTask As Task
Set oTask = oProj.Tasks.Add("Задача 1")
```

У объекта `Task` огромное количество свойств, которые позволяют настроить любые параметры задачи. Например, у этого объекта есть свойства `ActualStart`, `ActualFinish`, `ActualDuration` и т. п. Но догадаться, как каждое из этих свойств соотносится с нужным нам полем на графическом экране, очень сложно. Обычно намного удобнее после создания задачи использовать не ее свойства, а специальный метод `SetTaskField()` объекта `Application`. Отличие этого метода в том, что он принимает в качестве параметров имя свойства (на русском языке, как оно выглядит на графическом экране), значение и в качестве одного из необязательных параметров `Task.ID`. Например, начало и длительность созданной нами задачи можно установить так:

```
Application.SetTaskField "Начало", Date, False, False, oTask.ID
Application.SetTaskField "Длительность", "2д", False, False, oTask.ID
```

В принципе, средствами этого метода можно сразу создавать задачи, но это обычно не самый удобный способ.

Можно также после настройки нужного параметра на графическом экране сохранить проект в XML-файл и посмотреть информацию о настроенных параметрах элементов.

Другой важный элемент любого проекта — ресурсы. Работа с ресурсами включает в себя несколько задач.

Первая задача — это работа с корпоративным пулом ресурсов. Заполнение корпоративного пула, отслеживание изменений в нем и обеспечение актуальности — один из самых трудоемких компонентов работы с Project Server. Часто информацию о ресурсах нужно синхронизировать с внешними источниками данными (обычно с базами данных). Это можно сделать средствами Project Data Services, а можно — средствами VBA. Открыть корпоративный пул ресурсов можно так:

```
Application.EnterpriseResourcesOpen EUID:="", OpenType:=pjReadWrite
```

Если передать этому методу значения обоих параметров (они необязательные), то глобальный корпоративный пул ресурсов будет открыт без каких-либо диалоговых окон. Далее обычным способом (указанным ранее) ловим объектную ссылку на проект, имя которого — "Извлеченные корпоративные ресурсы", создаем в нем ресурсы, а потом сохраняем и закрываем. Другая возможность — импортировать ресурсы в глобальный корпоративный пул при помощи специального метода `Application.EnterpriseResourcesImport()`, которому передается `ID` локального ресурса.

Другая задача — создание локальных ресурсов. Здесь все просто. В объекте `Project` предусмотрена коллекция `Resources` с методом `Add()`, которая состоит из объектов `Resource`. Единственная проблема — то, что свойств у ресурсов тоже очень много. Но на помощь нам приходит метод `SetResourceField()`, аналогичный уже рассмотренному:

```
Dim oRes As Resource
Set oRes = oProj.Resources.Add("Иванов Иван")
Application.SetResourceField "Тип", "Трудовой", False, False, oRes.ID
```

Можно, конечно, использовать и стандартные свойства объекта:

```
oRes.StandardRate = 100
```

Разновидность этой задачи — поместить в локальный проект ресурсы из корпоративного пула. Для этой цели проще всего использовать метод `Application.EnterpriseResourceGet()`, которому нужно передать `ID` глобального ресурса и `ID` локального ресурса. Предварительно `ID` глобального ресур-

са можно получить, пройдя по нему циклом и выбрав нужные ресурсы по значению определенных полей.

Следующая задача — произвести назначения. Это можно сделать множеством разных способов:

- первый способ — воспользоваться коллекцией `Assignments` и ее метод `Add()`:

```
oTask.Assignments.Add oTask.ID, oRes.ID
```

- второй способ — применить метод `Application.ResourceAssignment()`. Он хорош тем, что позволяет назначать ресурсы одновременно нескольким задачам, но эти задачи должны быть выделены (пользователем или программным способом, что снижает надежность этого метода);
- третий способ — воспользоваться уже знакомым нам методом `SetTaskField()`:

```
SetTaskField Field:="Названия ресурсов", Value:="Иванов Иван[100%]"
```

Последнее, о чем нужно упомянуть, — это о назначении настраиваемых кодов структуры и корпоративных полей. Если обязательный настраиваемый корпоративный код структуры для проекта, задачи или ресурса не определен, то проект просто нельзя будет сохранить на сервере. Настраиваемые коды структуры и корпоративные поля настраиваются как обычные свойства. Если для задач и ресурсов они доступны напрямую (например, `oRes.EnterpriseText1 = "Мой текст"`), а для проекта — через свойство `SummaryTask`:

```
ActiveProject.ProjectSummaryTask.EnterpriseCost1 = "500.00"
```

Задание для самостоятельной работы 15: Программное создание проекта и его элементов

ЗАДАНИЕ:

Вам необходимо создать макрос, который программно создает проект "Забор" в файле Забор.mpp со следующими параметрами:

- время начала проекта — сегодняшнее число;
- задачи проекта:
  - вкапывание столбов. Исполнитель — Иванов Иван с почасовой ставкой 100 руб. в час. Время на выполнение — 2 рабочих дня (16 часов). Материальные ресурсы не используются;

- прибивание досок. Исполнитель — Петров Петр с почасовой ставкой 150 руб. в час. Время на выполнение — 2 рабочих дня (16 часов). Материальные ресурсы — пиломатериалы в размере 2 куб. м по цене 800 руб. за куб. м;
- покраска забора. Исполнитель — Сидорова Светлана с почасовой ставкой 125 руб. в час. Время на выполнение — 1 рабочий день (8 часов). Материальные ресурсы — краска в количестве 2 банки по цене 100 руб. за банку;

❐ каждую следующую задачу можно начинать только по завершении предыдущей.

Ответ к заданию 15

Код макроса, который можно запустить из Project, может быть таким:

```
Public Sub CreateProject()
    Dim oProject As Project
    Dim oTask As Task
    Dim oRes, oResMaterial As Resource
    Dim sID As String
    Dim oA As Assignment

    'Создаем новый проект
    Set oProject = Application.Projects.Add()

    'Создаем первую задачу и определяем дату начала и продолжительность
    Set oTask = oProject.Tasks.Add("Вкапывание столбов")
    Application.SetTaskField "Начало", Date, False, False, oTask.ID
    Application.SetTaskField "Длительность", "2д", False, False, oTask.ID

    'Создаем трудовой ресурс и настраиваем его свойства
    Set oRes = oProject.Resources.Add("Иванов Иван")
    oRes.Type = pjResourceTypeWork
    oRes.StandardRate = 100

    'Производим назначение ресурса задаче
    oTask.Assignments.Add oTask.ID, oRes.ID

    'Сохраняем идентификатор задачи — он потребуется,
    'чтобы потом указать эту задачу как предшественницу
    sID = oTask.ID

    'Создаем вторую задачу и второй трудовой ресурс
    Set oTask = oProject.Tasks.Add("Прибивание досок")
```

```
    oTask.Predecessors = sID
    Application.SetTaskField "Длительность", "2д", False, False, oTask.ID

    Set oRes = oProject.Resources.Add("Петров Петр")
    oRes.Type = pjResourceTypeWork
    oRes.StandardRate = 150

    'Создаем материальный ресурс и настраиваем его свойства
    Set oResMaterial = oProject.Resources.Add("Пиломатериалы")
    oResMaterial.Type = pjResourceTypeMaterial
    oResMaterial.StandardRate = 800
    oResMaterial.MaterialLabel = "куб. м"

    'Назначаем трудовой ресурс
    oTask.Assignments.Add oTask.ID, oRes.ID

    'Для материального ресурса нужно указать количество
    Set oA = oTask.Assignments.Add(oTask.ID, oResMaterial.ID)
    'Оно указывается ... в минутах!
    ' 120 минут (2 часа), т. е. 2 кубометра пиломатериалов
    oA.Work = 120

    'Опять сохраняем идентификатор задачи
    sID = oTask.ID

    'Для последней задачи выполняем уже знакомые нам действия
    Set oTask = oProject.Tasks.Add("Покраска забора")
    oTask.Predecessors = sID
    Application.SetTaskField "Длительность", "1д", False, False, oTask.ID

    Set oRes = oProject.Resources.Add("Сидорова Светлана")
    oRes.Type = pjResourceTypeWork
    oRes.StandardRate = 125

    Set oResMaterial = oProject.Resources.Add("Краска")
    oResMaterial.Type = pjResourceTypeMaterial
    oResMaterial.StandardRate = 100
    oResMaterial.MaterialLabel = "банка"

    oTask.Assignments.Add oTask.ID, oRes.ID

    Set oA = oTask.Assignments.Add(oTask.ID, oResMaterial.ID)
    oA.Work = 120

    'Сохраняем проект в соответствии с поставленными условиями
    oProject.SaveAs "C:\Забор.mpp"

End Sub
```