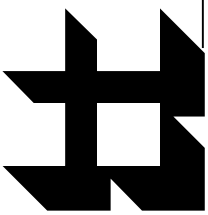


ГЛАВА 17

Автоматизация MS Office

Все приложения, входящие в пакет Microsoft Office, являются COM-серверами, поддерживающими как раннее, так и позднее связывание, поэтому выбор типа связи целиком ложится на клиентское приложение. В этой главе основной акцент сделан на использовании механизма позднего связывания, или автоматизацию. Мы рассмотрим здесь применение этой технологии для формирования отчетов в Microsoft Excel и Microsoft Word из приложений, написанных на Visual FoxPro.

Введение в Microsoft Office

Источником возможностей приложений Microsoft Office является наличие множества объектов, к которым вы можете обращаться из своих приложений. Эти объекты реализуют сложную иерархическую структуру, взаимодействие внутри которой описывается на специальном языке программирования VBA (Visual Basic for Application). Этот язык является упрощенной версией Visual Basic, и, что особенно важно, он ориентирован на применение внутри именно того приложения, составной частью которого является. Так, все свойства и методы объектов Microsoft Excel доступны из встроенного в него VBA; аналогично, все свойства и методы объектов Microsoft Word доступны из встроенного в него VBA.

Нужно отметить, что VBA не является монополией Microsoft Office; его используют и сторонние разработчики. Например, приложение AutoCAD, предназначенное для создания чертежей, так же имеет встроенный язык VBA.

VBA позволяет *программировать* документы. Создавая документ в приложении Microsoft Office, вы, как правило, сами управляете структурой этого документа, его форматированием, вводите, печатаете и сохраняете необходимые данные. Используя VBA, вы можете создавать специальные программы, или *макросы*, которые могут выполнять аналогичные действия по управлению структурой документа, его форматированием, вводу, печатью и сохранением данных.

Но чем может помочь VBA программисту Visual FoxPro, собирающемуся вывести отчет в Microsoft Excel? Собственно сам VBA — ничем, но вот написанный на нем

макрос помочь очень даже может, потому что он представляет собой последовательность команд, содержащих имена методов и свойств объектов, и правильный синтаксис операторов, обращающихся к ним. Если вы сумеете правильно интерпретировать код макроса при встраивании его в программу на Visual FoxPro, то проблема создания отчета будет вами успешно решена.

Объекты

При формировании отчета Visual FoxPro в приложении Microsoft Office обычно создается новый объект — экземпляр одного из классов этого приложения. Обычно для этого используются функции `CREATEOBJECT()` или `NEWOBJECT()`. Эти функции создают объект и возвращают ссылку на него:

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

или

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

Следует отметить, что объект `Application` лежит в основании объектной модели для любого приложения Microsoft Office. Но это не единственный объект верхнего уровня, который вы можете использовать. Например, вы можете создать объект — экземпляр класса рабочей книги Microsoft Excel:

```
oBook = CREATEOBJECT("Excel.WorkBook")
```

Список классов, предоставляемых Microsoft Excel, можно посмотреть в Обозревателе объектов Visual FoxPro (рис. 17.1).

Каждый из этих классов предоставляет свойство `Application`, возвращающее ссылку на объект `Application`:

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

Если вы хотите получить ссылку к уже существующему и выполняющемуся экземпляру сервера, то можете воспользоваться функцией `GETOBJECT()`.

Если функция пытается получить ссылку на объект, который не загружен, то генерируется исключение. В листинге 17.1 показан пример, в котором сначала делается попытка при помощи функции `GETOBJECT()` получить ссылку на уже выполняющийся сервер автоматизации; если попытка неудачна, то вызывается функция `CREATEOBJECT()` для создания нового объекта.

```
LOCAL loExcel, llError
TRY
   loExcel = GETOBJECT(, "Excel.Application")
CATCH                                          && Если ошибка, то
   loExcel = CREATEOBJECT("Excel.Application") && создать новый объект
```

```
ENDTRY
```

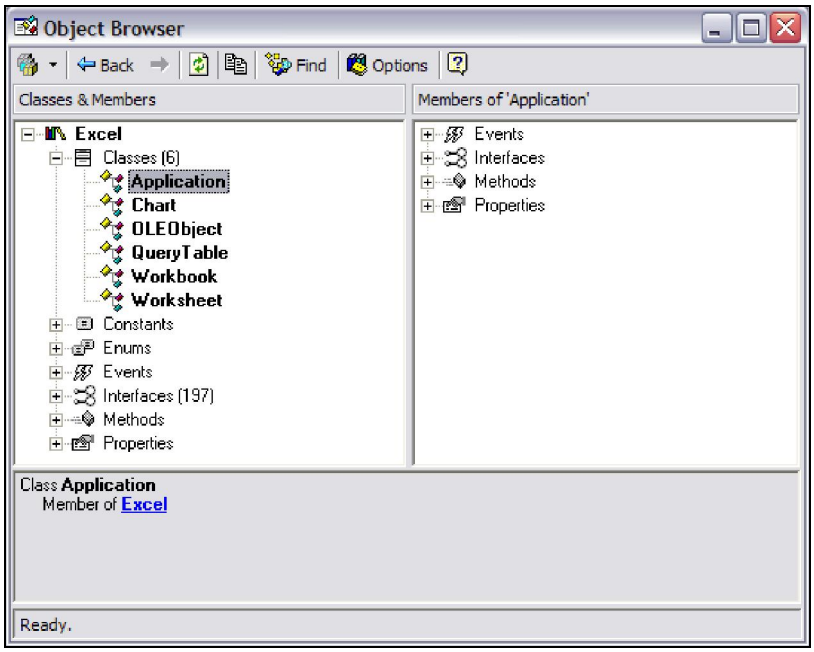


Рис. 17.1. Классы Microsoft Excel в Обозревателе объектов

Функция `GETOBJECT()` может использоваться для создания объекта и загрузки в него документа из файла:

```
loExcel = GETOBJECT("MyBook.xls")
```

или

```
loExcel = GETOBJECT("MyBook.xls", "Excel.Application")
```

В первом варианте выполняется поиск сервера, ассоциированного с расширением указанного файла, создается объект и в него загружается документ. Во втором варианте функции явно указывается `ProgID` сервера, в создаваемый экземпляр которого загружается документ из файла.

Для обращения к объекту внутри иерархии вы должны указать все предшествующие ему объекты. Вот как, например, может выглядеть команда записи данных в ячейку Microsoft Excel:

```
oExcel.WorkBook.WorkSheet.Cells(1,1).Value = "Текст"
```

Обращение к объекту внутри иерархии может быть упрощено за счет создания дополнительных ссылок и использования псевдонимов.

Использование раннего связывания

Встроенный в Visual FoxPro механизм IntelliSense отображает список всех свойств, методов и событий объекта в выпадающем списке, который формируется автоматически, как только вы ставите точку в коде листинга сразу за именем объекта. Для того чтобы этот механизм "работал" для COM-объектов, необходимо использовать раннее связывание.

На этапе разработки это делается достаточно просто: для переменной, которая будет использоваться как ссылка на COM-объект, явно указывается программный идентификатор этого объекта (ProgID):

```
LOCAL loExcel AS Excel.Application
```

Теперь IntelliSense "увидит" реквизиты объекта `Application` (рис. 17.2)

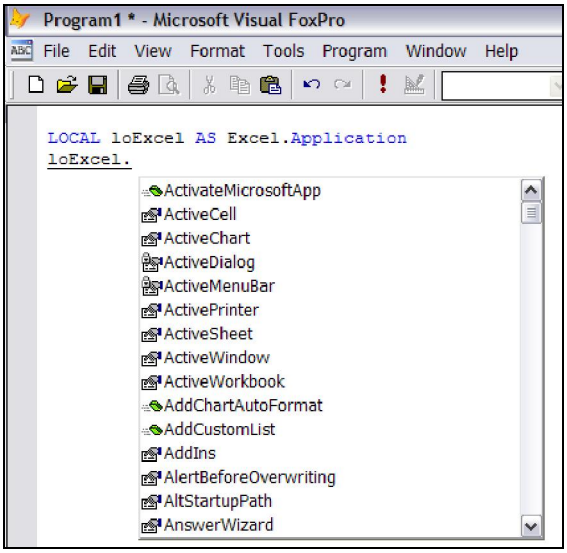


Рис. 17.2. Свойства и методы `Excel.Application` в окне IntelliSense

Как видите, нам даже не потребовалось создавать объект — экземпляр COM-класса, IntelliSense достаточно объявления типа переменной!

К сожалению, рассмотренная выше реализация раннего связывания действует только на этапе написания кода программы. При создании объекта при помощи функций `CREATEOBJECT()`, `NEWOBJECT()` или `GETOBJECT()` всегда реализуется позднее связывание.

Если вы хотите использовать раннее связывание на этапе выполнения приложения, то создавать COM-объекты нужно при помощи функции `CREATEOBJECTEX()`, как, например, в примере из *главы 16*:

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

Третий параметр, передаваемый функции, — это IID (уникальный идентификатор) интерфейса `_Application`, доступ к методам и свойствам которого вы хотите получить. Узнать IID для интерфейсов, предоставляемых сервером, вы можете при помощи Обозревателя объектов Visual FoxPro, загрузив в его нужную библиотеку типов. На рис. 17.3 в нижней области окна **Object Browser** вы можете видеть значение уникального идентификатора интерфейса `_Application`.

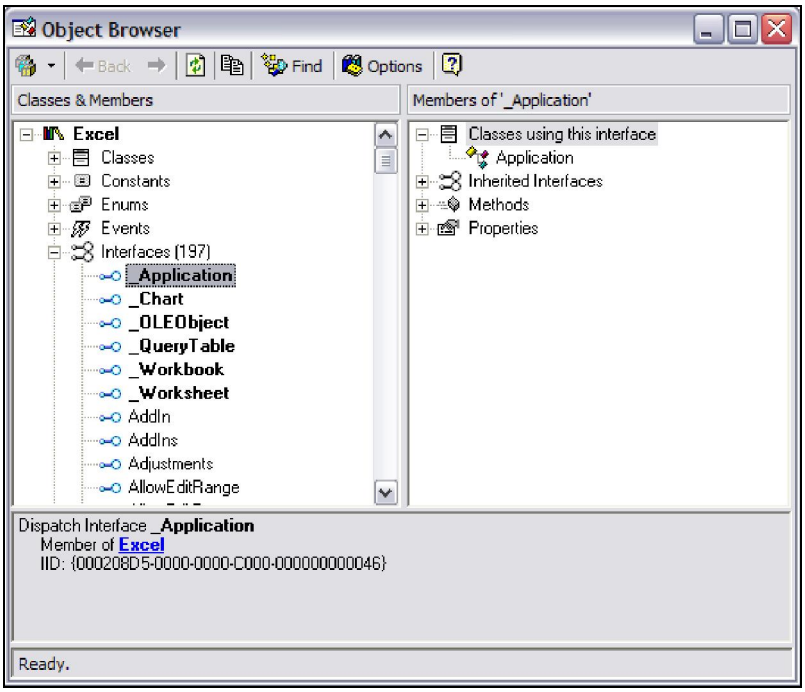


Рис. 17.3. IID интерфейса `_Application` в окне Обозревателя объектов

При использовании раннего связывания вы должны иметь в виду, что вам будут доступны только те свойства и методы, которые экспонируются выбранным интерфейсом.

Объекты-коллекции

Коллекции являются существенными элементами объектной модели Microsoft Office. Когда модели требуется иметь более одного экземпляра однотипных объектов, то для их отслеживания используются коллекции. Например, приложение Microsoft Excel может содержать более одной открытой рабочей книги, а каждая книга может иметь несколько рабочих листов. Таким образом, объект `Application` приложения Microsoft

Excel содержит коллекцию **WorkBooks**, в которой каждой открытой книге соответствует свой объект **WorkBook**. Каждый из объектов **WorkBook** содержит коллекцию **WorkSheets**, в которой находятся рабочие листы **WorkSheet**.

Каждый элемент в коллекции имеет числовой индекс, значения индекса могут изменяться от единицы до значения свойства `Count` коллекции, определяющего количество включенных в нее компонентов. Для обращения к элементу коллекции по его индексу используется свойство `Item`:

```
CollectionName.Item(nIndex)
```

Так как свойство `Item` является свойством, используемым объектами-коллекциями по умолчанию, то его можно опустить:

```
CollectionName(nIndex)
```

Обращаться к объектам в коллекции можно не только по индексу, но и по имени, или строке, идентифицирующей данный объект. Например, в коллекции `Documents` приложения Microsoft Word каждый объект `Document` для идентификации использует имя своего файла; аналогично, рабочий лист в коллекции **WorkSheets** может быть идентифицирован как по индексу, так и по наименованию:

```
Object.WorkSheets(3)          && Обращение по индексу
```

или

```
Object.WorkSheets("Лист 1")   && Обращение по имени
```

где *`Object`* — это ссылка на объект, содержащий коллекцию.

Макросы

Макрос — это фрагмент кода на VBA, содержащий программные инструкции для всех операций по вводу и форматированию данных, оформлению вида документа, его печати и сохранению в приложении Microsoft Office. Для записи макроса в меню **Сервис** приложения выберите пункт **Макрос** и затем, в появившемся справа дополнительном меню, пункт **Начать запись** (рис. 17.4).

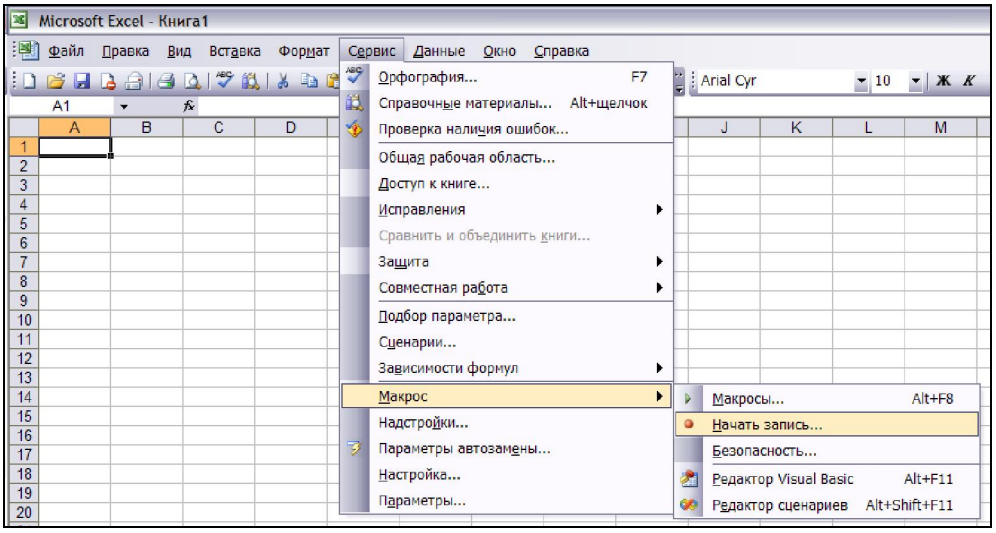


Рис. 17.4. Запись макроса в приложении Microsoft Excel

Перед тем как начать записывать все ваши действия по формированию документа как макроса, приложение выведет окно **Запись макроса**, позволяющее выполнить некоторые настройки. Это окно для каждого приложения Microsoft Office имеет свой специфический вид; вариант для Microsoft Excel показан на рис. 17.5, а для Microsoft Word — на рис. 17.6.

Рассмотрим шаги по созданию макроса на примере Microsoft Excel.

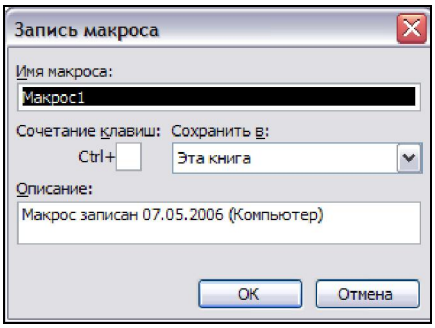


Рис. 17.5. Окно **Запись макроса** для Microsoft Excel

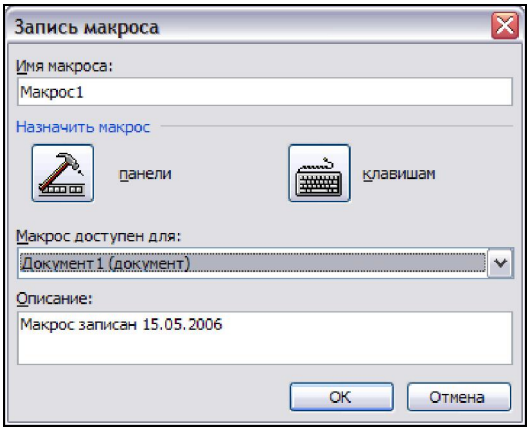


Рис. 17.6. Окно **Запись макроса** для Microsoft Word

Создание макроса в Microsoft Excel

В окне **Запись макроса** (рис. 17.5) вы можете изменить имя макроса и определить сочетание клавиш, при нажатии на которые макрос будет вызван на выполнение. Установите англоязычную языковую раскладку и введите в поле **Сочетание клавиш** символ "A". Теперь, после того как макрос будет сохранен, вы сможете запустить его на выполнение одновременным нажатием на клавиши <Ctrl>+<A>. Не изменяйте предложенное значение в поле **Сохранить в:** — макрос будет существовать только в пределах данной книги. Для перехода в режим записи макроса нажмите на кнопку **OK**. На экране появится маленькая панель инструментов, при помощи которой вы сможете остановить запись макроса.

Введите в таблицу данные, показанные на рис. 17.7 (в колонке **D** используйте формулы "`=B2*C2`" для второй строки и "`=B3*C3`" для третьей строки).

Остановите запись, выбрав в меню **Сервис** пункт **Макросы** и далее, в появившемся дополнительном меню, пункт **Остановить запись**, или нажмите на кнопку **Остановить запись** (на панели инструментов, показанной на рис. 17.5, она расположена слева). Выберите в меню пункты **Сервис | Макрос**. В появившемся меню выберите **Макросы...** (см. рис. 17.4). На экране появится окно **Макрос** (кстати, для вызова этого окна можно просто нажать клавиши <Alt>+<F8>).

В списке **Имя макроса** выберите созданный вами макрос (в нашем примере в этом списке всего один макрос с именем `Макрос1`) и нажмите на кнопку **Изменить**.

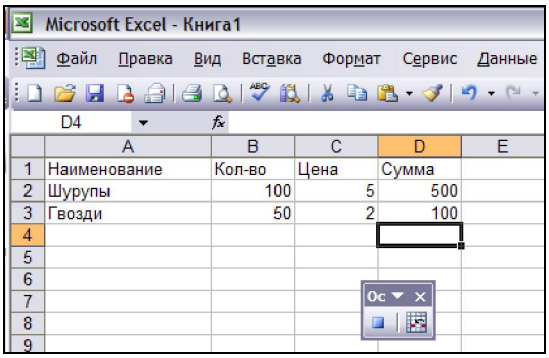


Рис. 17.7. Формирование макроса

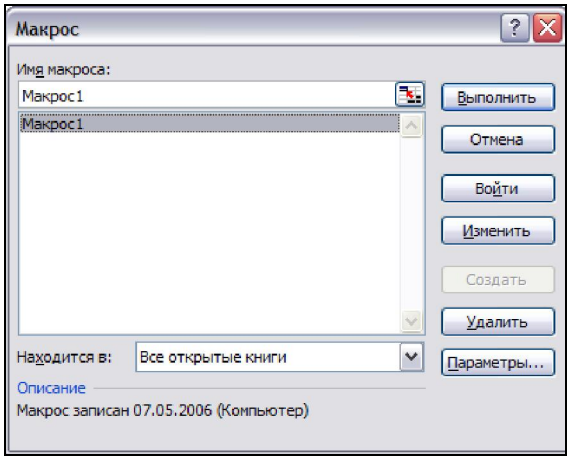


Рис. 17.8. Окно **Макрос**

Microsoft Excel загрузит редактор кода VBA (рис. 17.9), в котором будет выведен код сгенерированного макроса.

В листинге 17.2 показан код сгенерированного макроса VBA.

```
Sub Макрос1()
'
' Макрос1 Макрос
' Макрос записан 07.05.2006 (Компьютер)
'
' Сочетание клавиш: Ctrl+a
'
    Columns("A:A").ColumnWidth = 17.71
    ActiveCell.FormulaR1C1 = "Наименование"
    Range("B1").Select
    ActiveCell.FormulaR1C1 = "Кол-во"
    Range("C1").Select
    ActiveCell.FormulaR1C1 = "Цена"
    Range("D1").Select
    ActiveCell.FormulaR1C1 = "Сумма"
    Range("A2").Select
    ActiveCell.FormulaR1C1 = "Шурупы"
    Range("B2").Select
    ActiveCell.FormulaR1C1 = "100"
    Range("C2").Select
    ActiveCell.FormulaR1C1 = "5"
    Range("D2").Select
    ActiveCell.FormulaR1C1 = "=RC[-2]*RC[-1]"
    Range("A3").Select
```

```
    ActiveCell.FormulaR1C1 = "Гвозди"
    Range("B3").Select
    ActiveCell.FormulaR1C1 = "50"
    Range("C3").Select
    ActiveCell.FormulaR1C1 = "2"
    Range("D3").Select
    ActiveCell.FormulaR1C1 = "=RC[-2]*RC[-1]"
    Range("D4").Select
End Sub
```

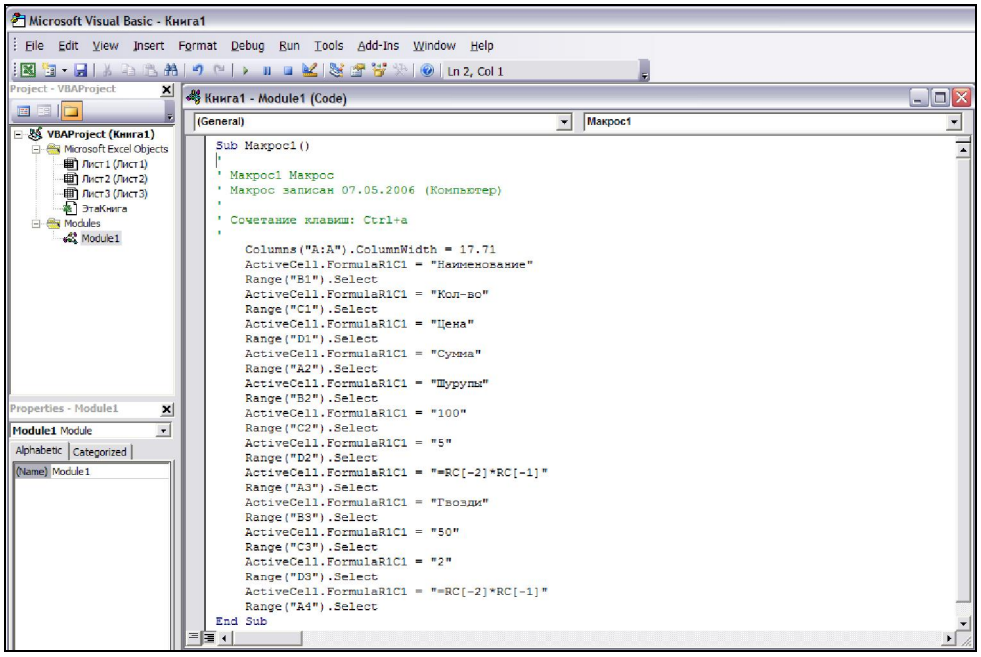


Рис. 17.9. Окно редактора кода VBA с кодом макроса

Используя буфер обмена Windows, сохраните полученный код в программном файле Visual FoxPro — он нам скоро понадобится.

Для проверки работоспособности созданного макроса перейдите на рабочий лист Microsoft Excel, сотрите созданную вами табличку и нажмите клавиши <Ctrl>+<A> для запуска макроса. Табличка должна восстановиться.

Выполнение макроса в Visual FoxPro

Измените код сохраненного в процедурном файле Visual FoxPro макроса так, как это показано в листинге 17.3.

```
LOCAL loExcel
loExcel = CREATEOBJECT("Excel.Application")
WITH loExcel
   .Visible = .t.       && Сделать окно Microsoft Excel видимым
   .Workbooks.Add       && Добавить новую книгу
   .Columns("A:A").ColumnWidth = 17.71
   .ActiveCell.FormulaR1C1 = "Наименование"
   .Range("B1").Select
   .ActiveCell.FormulaR1C1 = "Кол-во"
   .Range("C1").Select
   .ActiveCell.FormulaR1C1 = "Цена"
   .Range("D1").Select
   .ActiveCell.FormulaR1C1 = "Сумма"
   .Range("A2").Select
   .ActiveCell.FormulaR1C1 = "Шурупы"
   .Range("B2").Select
   .ActiveCell.FormulaR1C1 = "100"
   .Range("C2").Select
   .ActiveCell.FormulaR1C1 = "5"
   .Range("D2").Select
   .ActiveCell.FormulaR1C1 = "=RC[-2]*RC[-1]"
   .Range("A3").Select
   .ActiveCell.FormulaR1C1 = "Гвозди"
   .Range("B3").Select
   .ActiveCell.FormulaR1C1 = "50"
   .Range("C3").Select
   .ActiveCell.FormulaR1C1 = "2"
   .Range("D3").Select
   .ActiveCell.FormulaR1C1 = "=RC[-2]*RC[-1]"
   .Range("D4").Select
ENDWITH
```

Запустите процедуру на выполнение. Загрузится Microsoft Excel, и вы увидите на экране его окно, подобное показанному на рис. 17.7, но без панели инструментов для остановки записи макроса.

Как видите, код, представленный в листинге 17.3, весьма незначительно отличается от кода листинга 17.2. Добавлен вызов функции `CREATEOBJECT()` для создания объекта — экземпляра класса `Application` сервера Microsoft Excel; эта функция возвращает ссылку на созданный объект, которая сохраняется в переменной `loExcel`. Свойство `Visible` объекта `Application` устанавливается в "истину" — что заставляет Microsoft Excel нарисовать свое главное окно. Затем в коллекцию **WorkBooks** методом `Add` добавляется новая книга. А далее без каких-либо существенных изменений следует код макроса, рисующего таблицу; в соответствии с синтаксисом Visual FoxPro мы обращаемся ко всем перечисленным в коде макроса объектам, используя ссылку на объект `Application`.

Но если бы все было так просто, как в только что рассмотренном примере, то не было бы никакого смысла продолжать эту главу. На самом деле все гораздо сложнее, и на

пути формирования отчетов в приложениях Microsoft Office вас ожидает множество подводных камней. И в первую очередь это наличие в кодах макросов множества констант.

Определение значений констант

В коде макросов вы постоянно будете встречаться с константами. Значения констант присваиваются свойствам объектов; значения констант могут передаваться как параметры в методы объектов.

В Microsoft Excel константы имеют следующий вид:

```
xlИмя
```

где `xl` — это префикс, идентифицирующий константу. Примеры констант:

```
xlNone
xlTop
xlBottom
```

В Microsoft Word имя константы предваряется префиксом `wd`, например:

```
wdTableTop
wdSuffics
```

При использовании макроса в приложении Visual FoxPro вы должны вместо имен констант подставлять их значения. Поэтому проблема "а где их взять?" становится чрезвычайно актуальной.

Мы рассмотрим три способа получения значений констант:

- при помощи Обозревателя объектов Visual FoxPro;
- при помощи Обозревателя объектов Microsoft Office;
- при помощи макросов.

Просмотр констант в Обозревателе объектов Visual FoxPro

Запустите Обозреватель объектов Visual FoxPro и загрузите в него необходимую библиотеку типов; для Microsoft Excel библиотека типов включена в исполняемый модуль Excel.exe, а для Microsoft Word она находится в файле Msword.olb. Откройте узел **Constants** и установите указатель на одну из констант в списке (рис. 17.10).

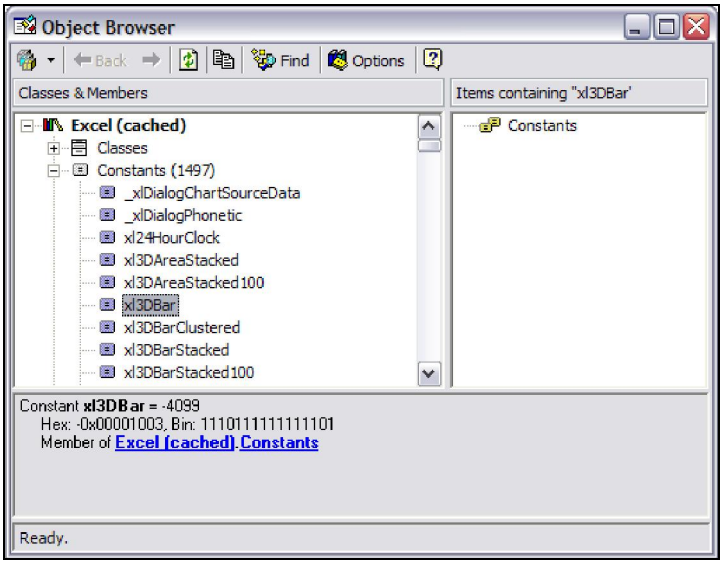


Рис. 17.10. Определение значений констант в Обозревателе объектов Visual FoxPro

В нижней области окна вы увидите десятичное, шестнадцатеричное и двоичное представление значения выбранной константы.

Просмотр констант в Обозревателе объектов Microsoft Office

Загрузите приложение Microsoft Office (например, Microsoft Excel) и перейдите в окно редактора VBA (для этого нужно нажать на клавиши <Alt>+<F11>). В меню **View** выберите пункт **Object Browser** (или нажмите на "горячую" клавишу <F2>). На экране появится окно **Object Browser** (рис. 17.11).

В расположенном слева вверху списке выберите библиотеку типов (Excel или Word — в зависимости от того, значение чьей константы вы хотите определить), а в расположенном под ним списке введите наименование константы. В области **Search Results** появится список найденных констант, удовлетворяющих условию запроса, в списке **Classes** будут выведены обозначения групп констант, используемых в классах, а в списке **Members of "имя_группы"** — сами константы; самое интересное, естественно, в нижней части окна: наименование константы и ее значение.

Определение значения константы при помощи макроса

Этот способ позволяет узнать значение конкретной константы. Создайте макрос и введите в него следующий код:

```
MsgBox имя_константы
```

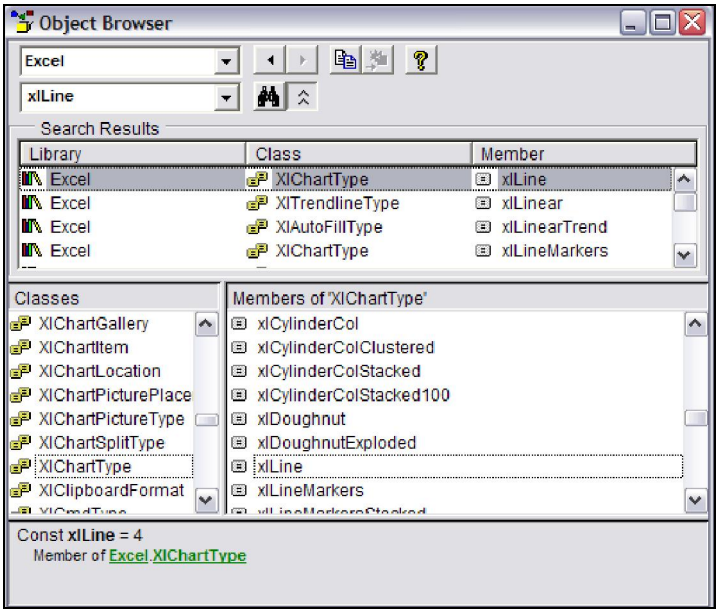


Рис. 17.11. Определение значений констант в Обозревателе объектов Microsoft Office

Запустите макрос на выполнение, например, нажав на кнопку **Выполнить** в окне Макрос (см. рис. 17.8). На экране появится окно, в котором будет выведено десятичное значение константы.

Константы Microsoft Office

Вы можете встретить в кодах макросов константы, имеющие префикс `mso`. Это константы уровня Microsoft Office. Для определения их значений загрузите в Обозреватель объектов Visual FoxPro библиотеку типов Microsoft Office 11.0 Object Library (для версии Microsoft Office, отличной от 11-й, эта строка будет содержать другой номер версии), или выберите в Обозревателе объектов Microsoft Office библиотеку Office. Далее действуйте, как описано выше.

На этом мы закончим общее знакомство с Microsoft Office и перейдем к рассмотрению его отдельных компонентов.

Автоматизация Microsoft Excel

Microsoft Excel — это приложение Microsoft Office, предназначенное для хранения, отображения и анализа данных в виде электронных таблиц и диаграмм. В нем реализовано множество специализированных функций для часто используемых вычислений, в том числе финансовых, статистических и научных. При создании приложений

на Visual FoxPro объекты Microsoft Excel можно использовать как для выполнения сложных расчетов, так и для формирования различных отчетов.

Объектная модель Microsoft Excel

Объектная модель Microsoft Excel имеет очень сложную иерархическую структуру, для описания всех компонентов которой потребовалась бы не одна толстая книга. Мы рассмотрим здесь только те компоненты этой модели, которые необходимы для формирования различных отчетов в виде таблиц и диаграмм.

Коллекция *WorkBooks* и объекты *WorkBook*

Коллекция **WorkBooks** объекта `Application` предназначена для хранения книг. Каждая книга представлена объектом `WorkBook`.

После создания объекта `Application` вы должны добавить в него книгу. Это может быть новая либо ранее созданная книга, сохраненная в файле с расширением xls.

Для создания новой книги используется метод `Add` коллекции **WorkBooks**:

```
oExcel = CREATEOBJECT("Excel.Application")
oBook = oExcel.WorkBooks.Add
```

Метод `Add` создает новый объект `WorkBook` и возвращает ссылку на него.

При создании новой книги вы можете использовать шаблоны (файлы с расширением xlt). Шаблон может включать в себя параметры форматирования, стили, стандартный текст, например, заголовки страниц и подписи строк и столбцов, формулы, макросы и пользовательские панели инструментов. Для создания книги с использованием шаблона нужно передать в метод `Add` имя файла шаблона:

```
oBook = oExcel.WorkBooks.Add(cTemplateFile)
```

где *`cTemplateFile`* — путь и имя файла шаблона.

> ***ЗАМЕЧАНИЕ***
>
> Когда Microsoft Excel запускается как обычное приложение, то в нем автоматически создается новая пустая книга. Если вы создаете объект — экземпляр класса сервера Microsoft Excel, то вы сами должны позаботиться о создании новой книги.

Метод `Open` создает новый объект `WorkBook`, загружая в него данные из указанного файла, и возвращает ссылку на созданный объект:

```
oExcel = CREATEOBJECT("Excel.Application")
oBook = oExcel.WorkBooks.Open(cFileName)
```

где *`cFileName`* — путь и имя открываемого файла.

Метод `Open` имеет большое число параметров, определяющих открытие файла только для чтения, задание пароля и многое другое. Подробно с синтаксисом этого метода вы можете ознакомиться в справочной документации по VBA.

Сохранение книги в файле выполняется методами `Save` и `SaveAs` объекта `WorkBook`. Метод `Save` не имеет аргументов; если книге уже было присвоено имя (или книга была ранее считана из существующего файла), то будет использовано это имя, иначе — Microsoft Excel попросит указать имя сохраняемого файла. Метод `SaveAs` позволяет сохранить новую книгу в указанном файле либо сохранить существующую книгу с новым именем:

```
oBook.Save()
```

или

```
oBook.SaveAs(cFileName)
```

Метод `SaveAs` так же имеет много дополнительных параметров, с которыми вы можете ознакомиться в справочной документации.

Для закрытия рабочей книги используется метод `Close` объекта `WorkBook`:

```
oBook.Close(SaveChanges, cFileName, RouteWorkBook)
```

Все три параметра не являются обязательными.

Параметр *`SaveChanges`* определяет действия в случае, когда в книге были сделаны изменения. Значением параметра является логическое выражение; "ложь" определяет закрытие книги без сохранения изменений. Значение параметра по умолчанию — "истина".

Параметр *`cFileName`* определяет путь и имя файла для сохранения книги.

Параметр *`RouteWorkBook`* относится только к тем рабочим книгам, которые имеют список распространения, но еще не были распространены.

Коллекции *WorkSheets* и *Sheets*

Коллекции **WorkSheets** и **Sheets** содержат объекты `WorkSheet` (рабочие листы), они доступны из объектов `Application` и `WorkBook`. Фактически имена этих коллекций являются синонимами, вы можете использовать любое из них по своему усмотрению.

При добавлении в коллекцию **WorkBooks** новой книги Microsoft Excel по умолчанию создает в ней три рабочих листа. Каждый лист в коллекции определяется индексом или именем. Изменить имя листа можно, установив свойство коллекции **Name**:

```
oBook.Sheets(1).name = "Первый лист"   && Устанавливаем новое имя листа
```

Методы `Select` и `Activate` коллекции позволяют перейти на любой лист книги, сделав его активным:

```
oBook.Sheets(3).Select
```

или

```
oBook.Sheets("Лист 3").Activate
```

ЗАМЕЧАНИЕ

Вы можете использовать для указания индекса или имени объекта в коллекции как круглые, так и квадратные скобки.

Свойство `Count` коллекции возвращает количество листов, включенных в книгу:

```
nCount = oBook.Sheets.Count
```

Метод `Add` добавляет в коллекцию новый объект `WorkSheet` и возвращает ссылку на него:

```
oSheet = oBook.Sheets.Add(Before, After, Count, Type)
```

Все четыре параметра метода являются необязательными. Если ни один из них не указан, то новый лист будет вставлен перед активным листом.

Параметры *`Before`* и *`After`* являются ссылками на рабочие листы. В следующем фрагменте кода показано, как вставить новый лист до или после третьего листа:

```
oBook.Sheets.Add(oBook.Sheets[3])   && Лист будет вставлен перед листом 3
```

или

```
oBook.Sheets.Add(,oBook.Sheets[3])  && Лист будет вставлен после листа 3
```

Параметр *`Count`* указывает количество вставляемых рабочих листов, а параметр *`Type`* определяет их вид. Возможные значения параметра *`Type`* приведены в табл. 17.1.

Таблица 17.1. *Значения параметра `Type` метода `Add` коллекции листов Excel*

Константа Excel	Значение	Описание
`xlChart`	-4109	Добавляет лист диаграммы
`xlDialogSheet`	-4116	Добавляет лист для создания диалогового окна (формы) Excel
`xlWorksheet`	-4167	Добавляет лист электронной таблицы (используется по умолчанию)

Удаление объекта `WorkSheet` из коллекции выполняет метод `Delete`. Удаляемый лист определяется либо его индексом в коллекции, либо именем:

```
oBook.Sheets[3].Delete
```

или

```
oBook.Sheets["Лист3"].Delete
```

Получить ссылку на рабочий лист в коллекции можно, указав его индекс или имя:

```
oSheet = oBook.Sheets[Item]
```

где *`Item`* — индекс или наименование листа.

Объект *WorkSheet*

Рабочие листы используются для формирования различных отчетов в виде электронных таблиц. Каждый такой лист состоит из коллекций колонок (Columns) и строк (Rows). На пересечениях строк и колонок находятся ячейки (Cell), образующие коллекцию **Cells**. На рабочих листах, кроме табличных данных, могут размещаться диаграммы.

Коллекции *Columns* и *Rows*

Коллекция **Columns** определяет группу ячеек, принадлежащих одной или нескольким подряд расположенным колонкам. Соответственно, коллекция **Rows** определяет группу ячеек, принадлежащих одной или несколько последовательно расположенным строкам. При обращении к элементу коллекции возвращается ссылка на объект `Range`, отображающий область ячеек.

Количество колонок на одном листе ограничено 255. Обычно колонки нумеруются символами латинского алфавита, начиная с A и заканчивая IV. Свойство `ColumnWidth` позволяет установить ширину для одной или нескольких подряд расположенных колонок по их индексам или наименованиям:

```
oSheet.Columns(15).ColumnWidth = 5.5    && Ширина для колонки № 15
oSheet.Columns("A:A").ColumnWidth = 50  && Ширина для колонки A
oSheet.Columns("B:E").ColumnWidth = 35  && Ширина для колонок B, C, D, E
```

Строки на рабочем листе нумеруются в порядке возрастания. Максимальное количество строк ограничено числом 65536. Вы можете программно устанавливать высоту для одной или группы строк, используя свойство `RowHeight`:

```
oSheet.Rows(25).RowHeight = 8.5         && Высота для строки № 25
oSheet.Rows("5:5").RowHeight = 12       && Высота для строки № 5
oSheet.Rows("7:9").RowHeight = 24       && Высота для строк №№ 7, 8, 9
```

Если вы хотите, чтобы ширина колонки или высота строки определялись автоматически на основе размеров отображаемых в ячейках данных, используйте метод `AutoFit`:

```
oSheet.Columns(интервал).AutoFit
oSheet.Rows(интервал).AutoFit
```

где *интервал* — допустимый способ задания интервала, например:

```
oSheet.Columns("B:E").AutoFit           && Автоподбор ширины колонок
```

> ***Замечание***
>
> Коллекции **Columns** и **Rows** так же доступны из объекта `Application`.

Форматирование текстовых строк

Для выравнивания строк по горизонтали и вертикали в колонках, строках и ячейках служат свойства `HorizontalAlignment` и `VerticalAlignment`. Допустимые значения для этих свойств приведены в табл. 17.2.

Таблица 17.2. *Константы, определяющие выравнивание строк*

Константа Excel	Значение	Описание
xlCenter	-4108	Центрирование
xlBottom	-4107	Выравнивание по вертикали по нижнему краю
xlTop	-4160	Выравнивание по вертикали по верхнему краю

Таблица 17.2 *(окончание)*

Константа Excel	Значение	Описание
xlLeft	-4131	Выравнивание по горизонтали по левому краю
xlRight	-4152	Выравнивание по горизонтали по правому краю
xlGeneral	1	Используется текущая установка

Свойство `WrapText` устанавливает режим переноса строк внутри ячейки. Если оно установлено в "истину", то перенос строк разрешен.

Свойство `Orientation` управляет направлением рисования текстовой строки. Оно может принимать значения от –90 до +90 (от вертикально вверх до вертикально вниз); значение по умолчанию — ноль.

Свойство `IndentLevel` определяет отступ от края ячейки при выводе текста.

В листинге 17.4 показан пример форматирования рабочего листа книги.

```
#DEFINE xlRight  -4152                  && Объявление констант Excel
#DEFINE xlCenter -4108
LOCAL loExcel, loBook, loSheet
loExcel = CREATEOBJECT("Excel.Application")
loExcel.Visible = .t.
loBook = loExcel.WorkBooks.Add
loSheet = loBook.Sheets[1]
WITH loSheet
   .Columns("A:A").ColumnWidth = 40      && Ширина колонки A
   .Columns("A:A").HorizontalAlignment = xlRight
   .Columns("B:E").ColumnWidth = 15      && Ширина колонок B, C, D и E
   .Columns("B:E").HorizontalAlignment = xlCenter
   WITH .Rows("1:1")                     && Для первой строки:
      .RowHeight = 27                    && высота 27 пунктов
      .HorizontalAlignment = xlCenter    && и центрирование по
      .VerticalAlignment = xlCenter      && горизонтали и вертикали
      .WrapText = .t.                    && Разрешить перенос строк
```

```
   ENDWITH
ENDWITH
```

Сохраните код примера в процедурном файле и запустите его на выполнение. Проверьте, что правила форматирования соответствуют установленным, причем выполненные позднее установки для первой строки отменяют ранее сделанные установки для колонок.

Использование свойства *Visible*

Это свойство доступно из объектов `Application`, `WorkSheet` и `Chart` (диаграмма). Установка свойства в "ложь" скрывает объект.

> ***ЗАМЕЧАНИЕ***
>
> Обратите внимание на использование свойства `Visible` для объекта `Application` в коде листинга 17.4. По умолчанию, после создания объекта он остается невидимым. Для того чтобы "заставить" COM-приложение нарисовать окно, это свойство нужно установить в "истину".

Ячейки

Объект `Cell` — это ячейка рабочего листа, именно то место, где хранятся и отображаются данные. Все ячейки рабочего листа образуют коллекцию **Cells**, которая доступна из объектов `Application` и `WorkSheet`.

Положение ячейки на рабочем листе определяется двумя координатами; первая координата определяет номер строки, а вторая — номер колонки. Целочисленные значения этих координат используются для идентификации ячейки в коллекции ячеек. Например, чтобы записать строку `Тест` в ячейку, расположенную по адресу `C5` (пятая строка, третья колонка), нужно выполнить следующий код:

```
Object.Cells(5,3).Value = "Тест"
```

Здесь *`Object`* — это ссылка на объект, которому принадлежит коллекция ячеек.

Для присваивания значения ячейке используются свойства `Value`, `Formula` и `FormulaR1C1`. Свойство `Value` можно так же использовать для получения значения ячейки:

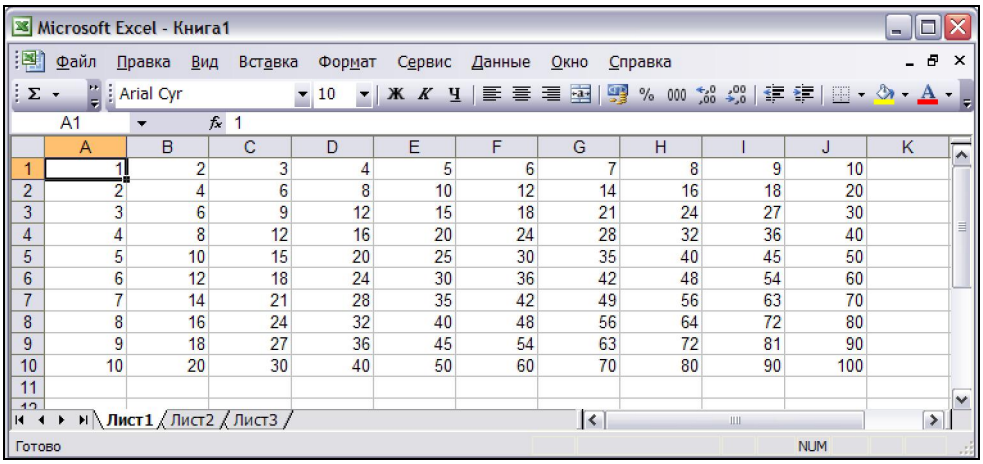
```
Contents = Object.Cells(5,3).Value
```

Для каждой ячейки можно установить уже рассмотренные выше параметры форматирования текста — `HorizontalAlignment`, `VerticalAlignment`, `Orientation` и `IndentLevel`; их действие будет распространяться только на эту ячейку.

В листинге 17.5 показан пример кода, демонстрирующий способ занесения данных в ячейку.

```
LOCAL loExcel, loBook, loSheet, lnRow, lnCol, lnValue
loExcel = CREATEOBJECT("Excel.Application")
loBook = loExcel.WorkBooks.Add
loSheet = loBook.Sheets[1]
WITH loSheet
   FOR lnRow = 1 TO 10
      FOR lnCol = 1 TO 10
         lnValue = lnRow * lnCol
         loSheet.Cells(lnRow,lnCol).Value = lnValue && Записать значение
      ENDFOR
   ENDFOR
ENDWITH
loExcel.Visible = .t.
```

Результат выполнения примера показан на рис. 17.12.



	A	B	C	D	E	F	G	H	I	J	K
1	1	2	3	4	5	6	7	8	9	10	
2	2	4	6	8	10	12	14	16	18	20	
3	3	6	9	12	15	18	21	24	27	30	
4	4	8	12	16	20	24	28	32	36	40	
5	5	10	15	20	25	30	35	40	45	50	
6	6	12	18	24	30	36	42	48	54	60	
7	7	14	21	28	35	42	49	56	63	70	
8	8	16	24	32	40	48	56	64	72	80	
9	9	18	27	36	45	54	63	72	81	90	
10	10	20	30	40	50	60	70	80	90	100	
11											

Рис. 17.12. Результат выполнения кода листинга 17.5

Шрифты

Для работы со шрифтами в VBA используется объект `Font`. В табл. 17.3 приведен список основных свойств этого объекта.

Таблица 17.3. *Свойства объекта* `Font`

Свойство	Значение	Описание
`Name`	Строка символов	Имя шрифта, например, "Arial"
`Size`	Число	Высота шрифта (в пунктах; пункт равен 1/72 дюйма)

`Bold`	Логическое	Полужирный шрифт
`Italic`	Логическое	Курсив
`Underline`	Число	Для подчеркивания текста присвойте свойству значение 2 (значение константы xlUnderlineStyleSingle)
`ColorIndex`	Число	Номер цвета в цветовой палитре (от 1 до 56)

В листинге 17.6 приведен модифицированный вариант кода листинга 17.5; в нем выполняется настройка стилей шрифта для колонок и цвета — для ячеек.

```
LOCAL loExcel, loBook, loSheet, lnRow, lnCol
loExcel = CREATEOBJECT("Excel.Application")
loExcel.Visible = .t.
loBook = loExcel.WorkBooks.Add
loSheet = loBook.Sheets[1]
WITH loSheet
   WITH .Columns("E:G").Font  && Шрифт для колонок E, F, G
      .Size = 14              && размер 14 пунктов
      .Name = "Courier New"   && Courier New
      .Italic = .t.           && курсив
   ENDWITH
   WITH .Columns("H:J").Font  && Шрифт для колонок H, I, J
      .Size = 12              && размер 12 пунктов
      .Name = "Georgia"       && Georgia
      .Bold = .t.             && полужирный
   ENDWITH
   FOR lnRow = 1 TO 10
      FOR lnCol = 1 TO 10
         loSheet.Cells(lnRow,lnCol).Value = lnRow * lnCol
* Изменение цвета выводимых строк
         loSheet.Cells(lnRow,lnCol).Font.ColorIndex = lnRow + lnCol
      ENDFOR
   ENDFOR
ENDWITH
```

Результат выполнения примера показан на рис. 17.13.

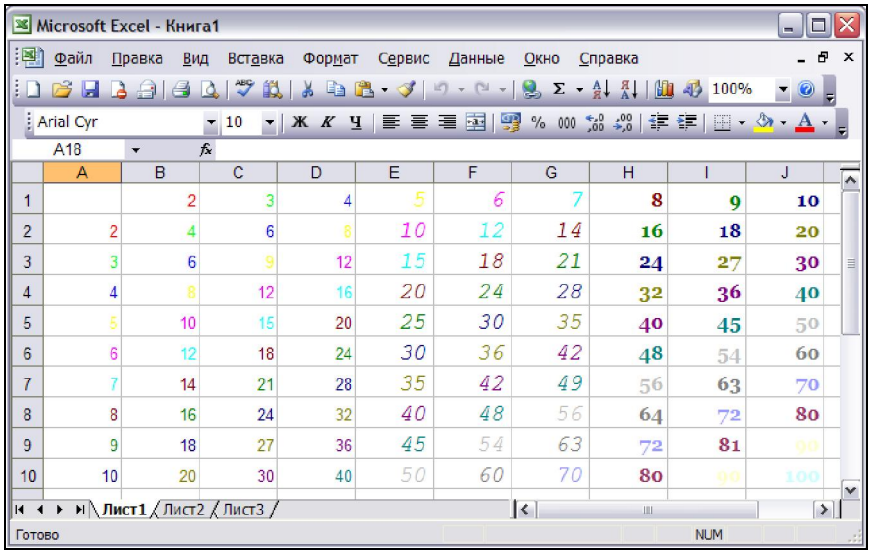


Рис. 17.13. Результат выполнения кода листинга 17.6

Не удивляйтесь тому, что часть значений не отображается в таблице — это потому, что значение `ColorIndex=2` определяет белый цвет.

Настройка шрифтов может быть выполнена для ячеек, колонок, строк и областей.

Коллекция *Range*. Области

Объект `Range` представляет собой коллекцию, в которой хранится от одной до множества ячеек. Совокупность этих ячеек образует область; вы можете устанавливать значения одновременно для всех свойств ячеек в области.

При задании прямоугольной области указываются имена ячеек, расположенных в верхнем левом и нижнем правом углу этой области. В следующем фрагменте кода выделяется и становится активной область, левый верхний угол которой определяется ячейкой `B3`, а правый нижний угол — ячейкой `F8`:

```
Object.Range("B3:F8").Select
```

здесь *Object* — ссылка на объект `Application` или `WorkSheet`.

Для задания координат области можно использовать коллекцию **Cells**:

```
Object.Range(Object.Cells(3,2),Object.Cells(8,6)).Select
```

А вот объявление области, состоящей из одной ячейки:

```
Object.Range("B3").Select
```

Вы можете использовать свойства `Value`, `Formula` и `FormulaR1C1` для присваивания одного значения всем ячейкам области:

```
Object.Range("B3:F8").Value = 0
```

Вы можете объединить включенные в область ячейки, установив в "истину" свойство `MergeCells` коллекции **Range**, и присвоить значение этой "объединенной" ячейке:

```
WITH Object.Range("B3:C4")
   .MergeCells = .t.
   .Value = "Текст"
ENDWITH
```

Вы так же можете форматировать отображаемые в ячейках данные, устанавливая для области значения свойств `HorizontalAlignment`, `VerticalAlignment`, `Orientation` и `IndentLevel`. А в следующем фрагменте кода показано, как установить параметры шрифта для всех ячеек области:

```
WITH Object.Range("B3:F8").Font
   .Size = 14                    && Установка высоты шрифта
   .Name = "Arial Cyr"           && Выбор типа шрифта
   .Italic = .T.                 && Начертание: курсив
ENDWITH
```

Внутри области используется собственная нумерация ячеек. Так, координаты ячейки, расположенной в верхнем левом углу области, будут равны (1,1).

Метод *Select* и свойство *Selection*

Вы уже встречали в рассматриваемых выше примерах ключевое слово `Select`, определяющее метод, делающий объект активным (активный объект имеет фокус ввода). При выполнении этого метода VBA создает свойство `Selection`, которое по своей функциональности можно сравнить с объектной ссылкой. Использования этого свойства демонстрируется в листинге 17.7.

```
#DEFINE xlRight  -4152          && Объявление констант
#DEFINE xlCenter -4108
LOCAL loExcel
loExcel = CREATEOBJECT("Excel.Application")
WITH loExcel
   .Visible = .t.
   .WorkBooks.Add
   .Columns("A:A").Select       && Создание ссылки
   WITH .Selection              && Использование ссылки для установки
      .ColumnWidth = 40         && свойств
      .HorizontalAlignment = xlRight
   ENDWITH
   .Columns("B:E").Select       && Создание ссылки
   WITH .Selection              && Использование ссылки
      .ColumnWidth = 15
```

```
      .HorizontalAlignment = xlCenter
   ENDWITH
   .Rows("1:1").Select          && Создание ссылки
   WITH .Selection              && Использование ссылки для установки
      .RowHeight = 27
      .HorizontalAlignment = xlCenter
      .VerticalAlignment = xlCenter
      .WrapText = .t.
   ENDWITH
ENDWITH
```

Присваивание значений. Типы и представление данных

Как правило, перед тем как писать код формирования отчета в Microsoft Excel, вы будете создавать макросы и затем "переводить" их на язык Visual FoxPro. Не пугайтесь того обилия команд, которые содержит макрос! В подавляющем случае вам будет достаточно команд, рассмотренных в этом разделе, остальные можно безболезненно проигнорировать. Тем не менее отдельно остановимся на свойствах, используемых для присваивания значений ячейкам (областям):

- `Value`. Это ключевое слово позволяет записывать в ячейку любые значения — число, строку, дату. VBA сам пытается определить тип данных присваиваемого значения.
- `Formula`. Это ключевое слово в основном предназначено для записи в ячейку результата вычисления формулы Microsoft Excel, но, применяя его, можно присваивать и иные значения.
- `FormulaR1C1`. Это ключевое слово позволяет обрабатывать смещения, или относительные адреса, для параметров (см. листинг 17.2). Вы так же можете использовать его для записи в ячейку любых значений.

Если вы хотите использовать в коде процедуры на Visual FoxPro формулы Microsoft Excel, то их необходимо включать в код как текстовые литералы, первым символом в которых является знак равенства (=):

```
Object.Cells(nRow, nCol).Formula = "=SUM(A1:D8)"
```

В Microsoft Excel можно явно указать тип отображаемых данных для выделенной области или ячейки. Для этого используется свойство `NumberFormat`. Например, следующий код устанавливает для ячейки числовой формат с двумя знаками после запятой и разделителями групп разрядов:

```
Object.Cells(nRow, nCol).NumberFormat = "#,##0.00"
```

А в следующем примере для области ячеек устанавливается строковый тип данных:

```
Object.Range("A14:C20").NumberFormat = "@"
```

Объект *Chart*

Рабочая книга Microsoft Excel имеет два способа отображения диаграмм.

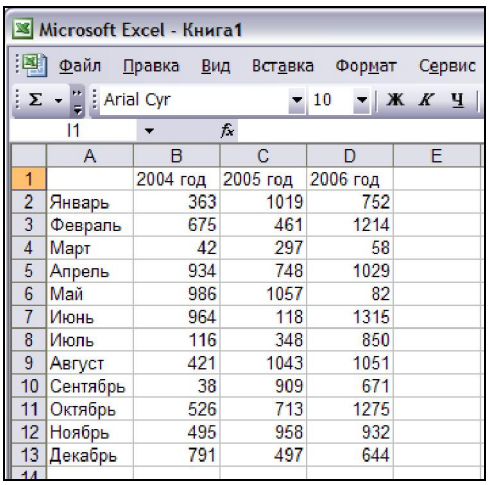
Первый способ отображает каждую диаграмму на отдельном листе, называемом листом диаграммы (Chart). Лист диаграммы содержит только одну диаграмму и не содержит колонок и строк с данными. Для хранения диаграмм книга использует коллекцию **Charts**.

Второй способ отображает диаграмму внедренной на рабочий лист. Каждая такая диаграмма также представлена объектом `Chart`, который существует не отдельно, а вложен в контейнер `ChartObject`; контейнер определяет положение и размеры диаграммы на листе. Рабочий лист поддерживает коллекцию **ChartObjects**, в которой содержится по одному объекту `ChartObject` для каждой внедренной на лист диаграммы.

Требования к исходным данным

Исходные данные, используемые для построения диаграммы, обычно организуются в виде прямоугольной таблицы, в которой самая левая колонка и самая верхняя строка содержат метки, идентифицирующие данные. Эта таблица размещается на одном из рабочих листов книги. На рис. 17.14 показан один из возможных вариантов подготовки исходных данных.

Для построения диаграммы объекту `Chart` необходимо передать ссылку на рабочий лист и выделить область ячеек, содержащих данные.



	A	B	C	D	E
1		2004 год	2005 год	2006 год	
2	Январь	363	1019	752	
3	Февраль	675	461	1214	
4	Март	42	297	58	
5	Апрель	934	748	1029	
6	Май	986	1057	82	
7	Июнь	964	118	1315	
8	Июль	116	348	850	
9	Август	421	1043	1051	
10	Сентябрь	38	909	671	
11	Октябрь	526	713	1275	
12	Ноябрь	495	958	932	
13	Декабрь	791	497	644	

Рис. 17.14. Пример исходных данных для построения диаграммы

Построение диаграммы на отдельном листе

Для построения диаграммы на отдельном листе нужно добавить лист диаграммы в коллекцию рабочих листов книги:

```
oChart = oBook.Sheets.Add(,,,xlChart)
```

В последней строке кода создается объект `Chart` и возвращается ссылка на него.

Определяем тип диаграммы, для чего присваиваем свойству `ChartType` численное значение, соответствующее одной из предопределенных констант Microsoft Excel (см. табл. 17.5):

```
oChart.ChartType = xlBarClustered
```

Записываем в свойство `SetSourceData` объекта ссылку на область с данными. Предположим, что рабочий лист с данными имеет индекс 2:

```
oChart.SetSourceData(oBook.Sheets[2].Range("A1:D12"), xlColumns)
```

Второй параметр, передаваемый методу, определяет, как будут организованы данные в серии: по строкам (`xlRows`; используется по умолчанию) или по столбцам (`xlColumns`).

Константы, определяющие способ построения диаграммы, перечислены в табл. 17.4, а константы, определяющей вид диаграммы — в табл. 17.5.

Таблица 17.4. *Константы, определяющие способ построения диаграммы*

Константа	Значение	Описание
`xlChart`	-4109	Определяет, что добавляемый лист является листом диаграммы. Передается как четвертый (последний) параметр в метод `Add` коллекции **Sheets**

Таблица 17.4 *(окончание)*

Константа	Значение	Описание
`xlRows`	1	Определяет, что данные в серии организованы по строкам
`xlColumns`	2	Определяет, что данные в серии организованы по столбцам

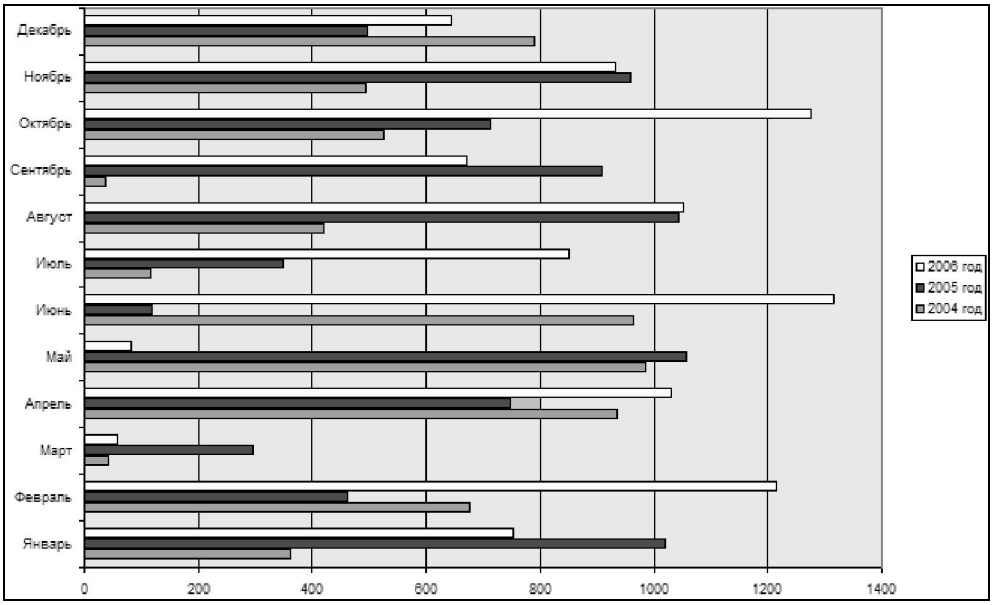


Рис. 17.15. Вид диаграммы, расположенной на отдельном листе

В листинге 17.8 приведен пример кода для построения диаграммы, а на рис. 17.15 показано, как эта диаграмма будет выглядеть.

```
#DEFINE xlChart      -4109
#DEFINE xlColumns        2
#DEFINE xlBarClustered  57
LOCAL i, j, loExcel, loBook, loChart, laMonth[12]
laMonth[1] = "Январь"
laMonth[2] = "Февраль"
laMonth[3] = "Март"
laMonth[4] = "Апрель"
laMonth[5] = "Май"
laMonth[6] = "Июнь"
laMonth[7] = "Июль"
laMonth[8] = "Август"
laMonth[9] = "Сентябрь"
laMonth[10] = "Октябрь"
laMonth[11] = "Ноябрь"
laMonth[12] = "Декабрь"
= RAND(-1)
loExcel = CREATEOBJECT("Excel.Application")
WITH loExcel
```

```
   .Visible = .t.
   loBook = .WorkBooks.Add
   loChart = .Sheets.Add(,,,xlChart)  && Добавляем лист диаграммы
   .Sheets[2].Select                  && Делаем активным рабочий лист
   .Cells(1,2).value = "2004 год"     && Формируем заголовки столбцов
   .Cells(1,3).value = "2005 год"
   .Cells(1,4).value = "2006 год"
   FOR i = 1 TO 12                    && Формируем значения столбцов
      j = i + 1
      .Cells(j,1).value = laMonth[i]
      .Cells(j,2).value = 1 + INT(1000 * RAND())
      .Cells(j,3).value = 1 + INT(1250 * RAND())
      .Cells(j,4).value = 1 + INT(1500 * RAND())
   ENDFOR
   WITH loChart
      .ChartType = xlBarClustered
      .SetSourceData(loBook.Sheets[2].Range("A1:D13"), xlColumns)
      .Activate                       && Делаем активным лист диаграммы
   ENDWITH
ENDWITH
```

Построение внедренной диаграммы

Как уже говорилось выше, для построения внедренной диаграммы мы должны создать коллекцию **ChartObjects**, и добавить в нее объект-контейнер `ChartObject` (или несколько таких объектов-контейнеров):

```
oChartObject = oExcel.ActiveSheet.ChartObjects.Add(X, Y, Width, Height)
```

В метод `Add` коллекции **ChartObjects** передаются координаты левой верхней точки, ширина и высота прямоугольной области, в которую будет помещена диаграмма; метод возвращает ссылку на добавленный в коллекцию объект-контейнер `ChartObject`. Теперь, используя эту ссылку, можно устанавливать свойства вложенного в него объекта `Chart`:

```
oChartObject.Chart.ChartType = xlBarClustered
```

На рис. 17.16 показана внедренная в рабочий лист диаграмма, а в листинге 17.9 приведен код для ее создания.

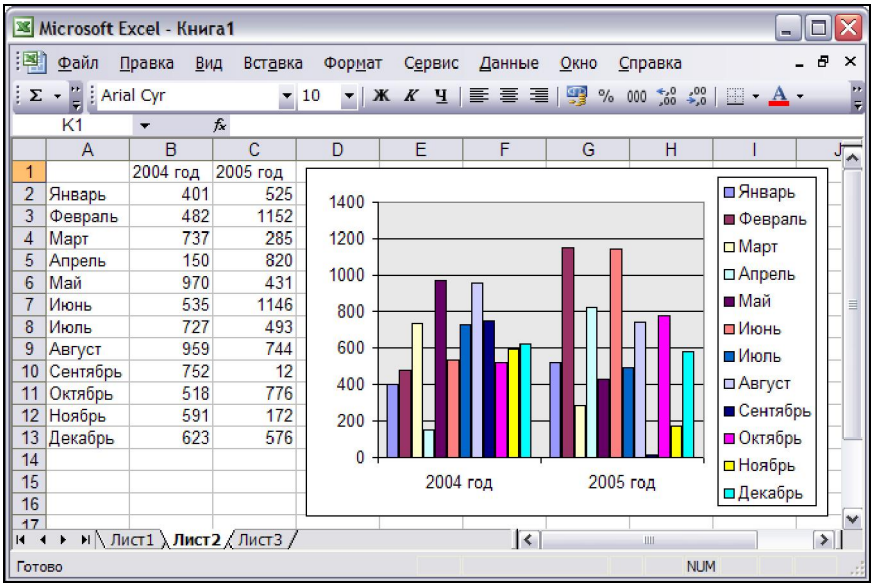


Рис. 17.16. Внедренная диаграмма

```
#DEFINE xlRows              1
#DEFINE xlColumnClustered  51
LOCAL i, j, loExcel, loChartObject, laMonth[12]
laMonth[1] = "Январь"
laMonth[2] = "Февраль"
laMonth[3] = "Март"
laMonth[4] = "Апрель"
laMonth[5] = "Май"
laMonth[6] = "Июнь"
laMonth[7] = "Июль"
laMonth[8] = "Август"
laMonth[9] = "Сентябрь"
laMonth[10] = "Октябрь"
laMonth[11] = "Ноябрь"
laMonth[12] = "Декабрь"
= RAND(-1)
loExcel = CREATEOBJECT("Excel.Application")
WITH loExcel
   .Visible = .t.
   .WorkBooks.Add
   .Cells(1,2).value = "2004 год"       && Формируем заголовки столбцов
   .Cells(1,3).value = "2005 год"
```

```
   FOR i = 1 TO 12                    && Формируем значения столбцов
      j = i + 1
      .Cells(j,1).value = laMonth[i]
      .Cells(j,2).value = 1 + INT(1000 * RAND())
      .Cells(j,3).value = 1 + INT(1250 * RAND())
   ENDFOR
   loChartObject = .ActiveSheet.ChartObjects.Add(150,5,300,200)
   WITH loChartObject.Chart
      .ChartType = xlColumnClustered
      .SetSourceData(loExcel.Range("A1:C13"), xlRows)
   ENDWITH
ENDWITH
```

Обратите внимание на использование в коде псевдонима `ActiveSheet` для ссылки на активный лист.

Виды диаграмм

В табл. 17.5 перечислены некоторые константы Microsoft Excel, определяющие виды диаграмм (гистограмма, линейчатая, график, круговая и т. д.).

Таблица 17.5. *Значения констант Excel, определяющих типы диаграмм*

Константа	Значение	Константа	Значение
xlArea	1	xlLineStacked100	64
xlLine	4	xlLineMarkersStacked	66
xlPie	5	xlLineMarkersStacked100	67
xlBubble	15	xlPieOfPie	68
xlColumnClustered	51	xlPieExploded	69
xlColumnStacked	52	xl3DPieExploded	70
xlColumnStacked100	53	xlBarOfPie	71
xl3DColumnClustered	54	xlAreaStacked	78
xl3DColumnStacked	55	xlAreaStacked100	79
xl3DColumnStacked100	56	xl3DAreaStacked	80
xlBarClustered	57	xl3DAreaStacked100	81
xlBarStacked	58	xlRadarFilled	82
xlBarStacked100	59	xlSurface	83
xl3DBarClustered	60	xlSurfaceWireframe	84
xl3DBarStacked	61	xlSurfaceTopView	85
xl3DBarStacked100	62	xlSurfaceTopViewWireframe	86

xlLineStacked	63	xlBubble3DEffect	87

Таблица 17.5 (окончание)

Константа	Значение	Константа	Значение
xlCylinderColClustered	92	xlConeBarClustered	102
xlCylinderColStacked	93	xlConeBarStacked	103
xlCylinderColStacked100	94	xlConeBarStacked100	104
xlCylinderBarClustered	95	xlPyramidColClustered	106
xlCylinderBarStacked	96	xlPyramidColStacked	107
xlCylinderBarStacked100	97	xlPyramidColStacked100	108
xlConeColClustered	99	xlPyramidBarClustered	109
xlConeColStacked	100	xlPyramidBarStacked	110
xlConeColStacked100	101	xlPyramidBarStacked100	111

Управление внешним видом диаграммы

Объект `Chart` предоставляет большой набор свойств и методов, позволяющих изменять внешний вид диаграммы.

Если вы хотите снабдить свою диаграмму заголовком, то необходимо установить следующие свойства:

```
oChart.HasTitle = .t.                && Разрешить вывод заголовка
oChart.ChartTitle.Characters.Text = "Текст заголовка"
```

Для изменения шрифта заголовка используйте объект `Font`:

```
WITH oChart.ChartTitle
   .AutoScaleFont = .t.       && Разрешить автоматическое масштабирование
   .Font.Name  = "Times New Roman"
   .Font.FontStyle = "полужирный"
   .Font.Size = 12
ENDWITH
```

По умолчанию выводится легенда диаграммы. Вы можете запретить вывод легенды, присвоив значение "ложь" свойству `HasLegend`:

```
oChart.HasLegend = .f.
```

Вы можете изменить расположение легенды, разместив ее сверху, внизу, справа или слева от диаграммы.

```
oChart.HasLegend = .t.    && Разрешить вывод легенды
oChart.Legend.Position = xlBottom
```

Значение константы `xlBottom`, а также других констант, определяющих позицию, см. в табл. 17.2.

Используйте объект `Font` для изменения шрифта, применяемого в легенде:

```
WITH oChart.Legend
   .AutoScaleFont = .t.   && Разрешить автоматическое масштабирование
   .Font.Name  = "Arial Cyr"
   .Font.FontStyle = "курсив"
   .Font.Size = 10
ENDWITH
```

При помощи свойства `PlotArea` вы можете установить цвет фона и параметры рамки, обрамляющей диаграмму:

```
WITH oChart.PlotArea
   .Border.ColorIndex = 4      && Цвет рамки
   .Border.Weight = -4138      && xlMedium — линия сред. толщины
   .Interior.ColorIndex = 3    && Цвет фона
   .Interior.Pattern = 1       && xlSolid — тип кисти
ENDWITH
```

Свойство `Border` определяет параметры рамки, а свойство `Interior` — параметры фона. Обратите внимание на определение цвета при помощи свойства `Color`. Вы можете вместо этого свойства использовать `ColorIndex`; в случае выбора цвета при помощи функции RGB вы должны помнить, что значения компонентов цветовой палитры должны изменяться с шагом, кратным 32.

Все элементы диаграммы объединены в серии, которые образуют коллекцию **SeriesCollection**. Ниже показано, как изменить параметры третьего элемента серии:

```
oChart.SeriesCollection(3).Select
WITH .Selection
   .Interior.ColorIndex = 33  && Цвет элемента
   .Interior.Pattern = 1      && xlSolid — тип кисти
   .Shadow = .t.              && Разрешить прорисовку тени элемента
ENDWITH
```

Мы рассмотрели лишь некоторые из возможностей объекта `Chart`. Если вы хотите использовать иные возможности объекта, то лучший способ для этого — создание и последующий анализ макросов. Вот все, что нужно сделать для этого:

- сформируйте на рабочем листе таблицу с исходными данными для построения диаграммы;
- запустите **Мастер диаграмм** (меню **Вставка** | **Диаграмма**);
- включите запись макроса;
- щелкая по областям созданной диаграммы, в интерактивном режиме выполните необходимые настройки внешнего вида этих областей;
- завершите запись макроса и проанализируйте полученный код.

Оформление отчетов

Основные приемы форматирования вам уже известны. Вы умеете программно объединять ячейки, форматировать текст и настраивать шрифты. Но этого явно недостаточно для оформления отчетов; вы должны уметь программно обводить табличные данные рамкой (как вы обычно делаете, форматируя документ вручную), выделять цветом определенные области документа и, наконец, вставлять в отчет изображения.

Рисование линий и рамок вокруг ячеек и областей

Для того чтобы нарисовать рамку вокруг выделенной области, необходимо последовательно нарисовать все линии, обрамляющие эту область. Область может состоять как из одной ячейки, так и из группы ячеек.

Коллекция **Borders** объекта `Range` содержит объекты `Border`, каждый из которых отвечает за рисование линии по границе или внутри области. Константы, определяющие индексы объектов `Border` в коллекции, перечислены в табл. 17.6.

Таблица 17.6. *Константы, идентифицирующие объекты* `Border` *в коллекции*

Константа	Значение	Описание
`xlDiagonalDown`	5	Рисует линию, соединяющую левый верхний и правый нижний углы каждой ячейки области
`xlDiagonalUp`	6	Рисует линию, соединяющую левый нижний и правый верхний углы каждой ячейки области
`xlEdgeLeft`	7	Рисует линию по левой границе области
`xlEdgeTop`	8	Рисует линию по верхней границе области
`xlEdgeBottom`	9	Рисует линию по нижней границе области
`xlEdgeRight`	10	Рисует линию по правой границе области
`xlInsideVertical`	11	Рисует вертикальные линии внутри области по границам ячеек
`xlInsideHorizontal`	12	Рисует горизонтальные линии внутри области по границам ячеек

Свойство `LineStyle` определяет стиль линии. Его допустимые значения перечислены в табл.17.7.

Таблица 17.7. *Константы, определяющие значение свойства* `LineStyle` *объекта* `Border`

Константа	Значение	Описание
`xlNone`	-4142	Линия не рисуется

xlContinuous	1	Непрерывная линия (используется по умолчанию)
xlDash	-4115	Пунктирная линия
xlDashDot	4	Штрихпунктирная линия
xlDashDotDot	5	Штрих-штрихпунктирная линия

***Таблица 17.7** (окончание)*

Константа	Значение	Описание
xlDot	-4118	Штриховая линия
xlDouble	-4119	Двойная линия

Свойство `Color` определяет цвет линии в формате RGB, а свойство `ColorIndex` — по индексу в палитре цветов.

Свойство `Weight` определяет толщину линии. Его допустимые значения перечислены в табл. 17.8.

***Таблица 17.8.** Константы, определяющие значение свойства `Weight` объекта `Border`*

Константа	Значение	Описание
xlHairline	1	Очень тонкая линия
xlThin	2	Тонкая линия
xlMedium	-4138	Линия средней толщины
xlThick	4	Толстая линия

В листинге 17.10 приведен пример кода, рисующий рамку вокруг выделенной области синей линией средней толщины и заполняющий эту область вертикальными и горизонтальными тонкими пунктирными линиями серого цвета.

```
#DEFINE xlEdgeLeft           7
#DEFINE xlEdgeTop            8
#DEFINE xlEdgeBottom         9
#DEFINE xlEdgeRight         10
#DEFINE xlInsideVertical    11
#DEFINE xlInsideHorizontal  12
#DEFINE xlThin               2
#DEFINE xlMedium         -4138
#DEFINE xlDot            -4118
LOCAL loExcel, lnBorderColor, lnLineColor
lnBorderColor = RGB(0,0,128)
lnLineColor = RGB(192,192,192)
```

```
loExcel = CREATEOBJECT("Excel.Application") && Создаем объект
WITH loExcel
   .Visible = .t.                          && Делаем окно Excel видимым
   .WorkBooks.Add                          && Добавляем рабочую книгу
   .Range("B2:D8").Select                  && Выделить область ячеек A2-D8
   WITH .Selection                         && Для выделенной области:
      WITH .Borders(xlEdgeLeft)            && Левую границу рисуем
         .Weight = xlMedium                && линией средней толщины
         .Color = lnBorderColor            && синего цвета
      ENDWITH
      WITH .Borders(xlEdgeTop)             && Верхнюю границу рисуем
         .Weight = xlMedium                && линией средней толщины
         .Color = lnBorderColor            && синего цвета
      ENDWITH
      WITH .Borders(xlEdgeBottom)          && Нижнюю границу рисуем
         .Weight = xlMedium                && линией средней толщины
         .Color = lnBorderColor            && синего цвета
      ENDWITH
      WITH .Borders(xlEdgeRight)           && Правую границу рисуем
         .Weight = xlMedium                && линией средней толщины
         .Color = lnBorderColor            && синего цвета
      ENDWITH
      WITH .Borders(xlInsideVertical)      && Вертикальные линии внутри
         .Weight = xlThin                  && области рисуем тонкой
         .Color = lnLineColor              && пунктирной линией
         .LineStyle = xlDot                && серого цвета
      ENDWITH
      WITH .Borders(xlInsideHorizontal)    && Так же выглядят и
         .Weight = xlThin                  && горизонтальные линии
         .Color = lnLineColor              && внутри области
         .LineStyle = xlDot
      ENDWITH
   ENDWITH
ENDWITH
```

Результат выполнения кода показан на рис. 17.17.

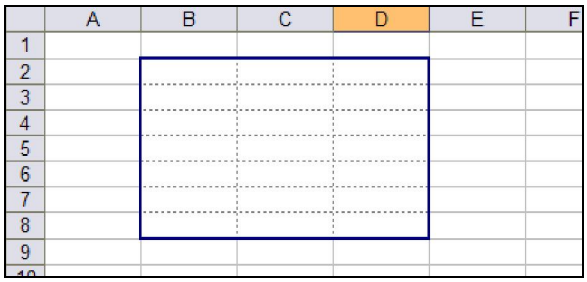


Рис. 17.17. Результат выполнения кода листинга 17.13

Заливка ячеек и областей

Вы уже применяли сплошную кисть при заливке элементов диаграммы. Помимо сплошной, возможно использование и других кистей, например, штриховых или градиентных.

Фон определяется свойством `Interior` ячейки (области), тип кисти указывается в свойстве `Pattern`, а цвет — в свойстве `PatternColor` (для RGB-палитры) или `PatternColorIndex`.

Следующий фрагмент кода выполняет заливку ячейки желтым цветом:

```
WITH Object.Cells(nRow, nCol).Interior
   .Pattern = 1                    && xlSolid: сплошная кисть
   .Color = RGB(255,255,0)
ENDWITH
```

А в следующем фрагменте кода выполняется заливка области желтым цветом и штриховка кистью синего цвета:

```
WITH Object.Range(("A1:F10").Interior
   .Pattern = 15                   && xlPatternGrid: прямоугольная сетка
   .Color = RGB(255,255,0)
   .PatternColor = RGB(0,0,255)
ENDWITH
```

Использование градиентных кистей требует гораздо больших затрат по написанию кода. Полагаем, что, при необходимости, вы научитесь пользоваться и такими кистями, создав несколько специальных макросов и проанализировав их код.

Вставка в отчет рисунков

Вы можете разместить на рабочем листе любой рисунок, определив его положение и размеры. Следующая команда создает объект `Shape` и загружает в него изображение:

```
oShape = oSheet.Pictures.Insert(cFileName)
```

Здесь *oSheet* — ссылка на рабочий лист, а `oShape` — ссылка на объект *Shape*, в который загружено изображение из файла, указанного в *cFileName*. Используя свойства *Top*, *Left*, *Width* и *Height* объекта *Shape*, вы можете поместить изображение в любую позицию рабочего листа и установить его размеры. В качестве единицы измерения применяются пикселы.

В листинге 17.11 показан пример кода для вставки изображения на рабочий лист.

```
LOCAL lcFileName, loExcel, loShape
lcFileName = GETFILE("BMP|JPG|GIF|PNG|TIF")
IF !EMPTY(m.lcFileName)
```

```
   loExcel = CREATEOBJECT("Excel.Application")
   WITH loExcel
      .Visible = .t.
      .WorkBooks.Add
      loShape = .ActiveSheet.Pictures.Insert(lcFileName) && Загрузка
   ENDWITH
   WITH loShape
      .Top = 100       && Координата X верхнего левого угла
      .Left = 10       && Координата Y верхнего левого угла
      .Width = 350     && Ширина
      .Height = 250    && Высота
   ENDWITH
ENDIF
```

Вы можете повернуть размещенное на листе изображение на произвольный угол:

```
oShape.IncrementRotation(nAngle)
```

где *oShape* — ссылка на объект `Shape`, а *nAngle* — угол поворота в градусах.

Цветовая палитра

Завершая тему оформления отчетов, познакомимся еще с одной коллекцией, принадлежащей рабочей книге, — коллекцией цветов **Colors**. Эта коллекция содержит 56 элементов, образующих палитру из 56 цветов. Когда вы присваиваете значение индекса свойству `ColorIndex` объекта, тем самым вы определяете, из какого элемента коллекции будет взят нужный цвет. К сожалению, нет возможности изменить количество используемых в книге Microsoft Excel цветов; но вы можете изменить цвет любого элемента коллекции, как, например, в следующем фрагменте кода:

```
oBook.Colors(5) = RGB(255,255,232)
```

Если на рабочих листах книги уже есть окрашенные элементы, то при внесении изменений в коллекцию **Colors** их цвета будут изменены.

Формирование отчетов

Для рассмотрения формирования табличных отчетов в Microsoft Excel нам понадобятся три таблицы:

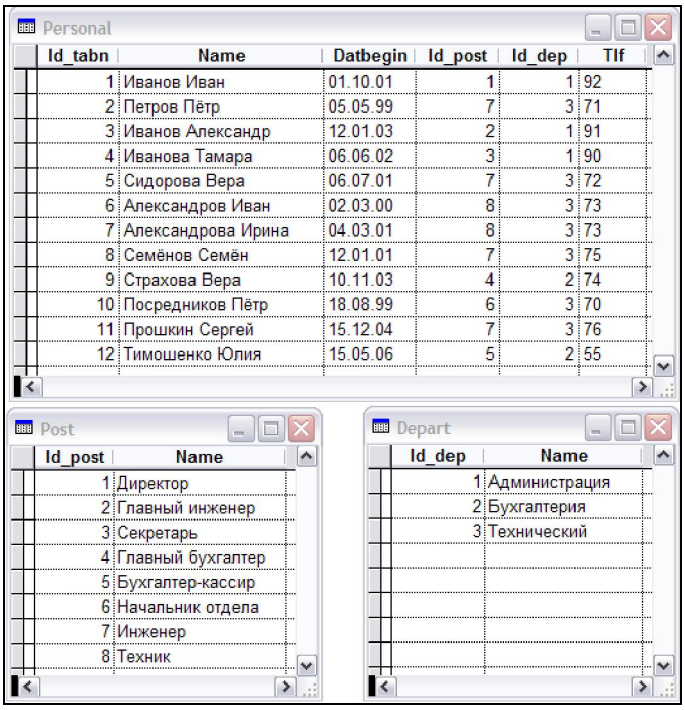
- таблица **Personal**; содержит список сотрудников;
- таблица **Depart**; содержит список подразделений;
- таблица **Post**; содержит список должностей.

Структура этих таблиц и хранимые в них данные показаны на рис. 17.18.

На основании этих таблиц мы сформируем в Microsoft Excel два различных отчета. Первый отчет будет представлять собой плоскую таблицу, содержащую все данные из базы данных. Второй отчет будет создан в виде иерархической таблицы, данные в которой будут сгруппированы по подразделениям и должностям.

Формирование отчета в виде плоской таблицы

Код, формирующий такой отчет, представлен в листинге 17.12.



Personal

Id_tabn	Name	Datbegin	Id_post	Id_dep	Tlf
1	Иванов Иван	01.10.01	1	1	92
2	Петров Пётр	05.05.99	7	3	71
3	Иванов Александр	12.01.03	2	1	91
4	Иванова Тамара	06.06.02	3	1	90
5	Сидорова Вера	06.07.01	7	3	72
6	Александров Иван	02.03.00	8	3	73
7	Александрова Ирина	04.03.01	8	3	73
8	Семёнов Семён	12.01.01	7	3	75
9	Страхова Вера	10.11.03	4	2	74
10	Посредников Пётр	18.08.99	6	3	70
11	Прошкин Сергей	15.12.04	7	3	76
12	Тимошенко Юлия	15.05.06	5	2	55

Post

Id_post	Name
1	Директор
2	Главный инженер
3	Секретарь
4	Главный бухгалтер
5	Бухгалтер-кассир
6	Начальник отдела
7	Инженер
8	Техник

Depart

Id_dep	Name
1	Администрация
2	Бухгалтерия
3	Технический

Рис. 17.18. Таблицы **Personal**, **Post** и **Depart**

```
#DEFINE xlCenter         -4108     && Описание констант Microsoft Excel
#DEFINE xlSolid              1
#DEFINE xlEdgeLeft           7
#DEFINE xlEdgeTop            8
#DEFINE xlEdgeBottom         9
#DEFINE xlEdgeRight         10
#DEFINE xlInsideVertical    11
#DEFINE xlInsideHorizontal  12
#DEFINE xlThin               2
```

```
#DEFINE xlThick             4
#DEFINE xlMedium        -4138
#DEFINE xlDouble        -4119
LOCAL lcPath, loExcel, lnRow
lcPath = JUSTPATH(SYS(16))
SET DEFAULT TO (lcPath)
CLOSE TABLES ALL
USE personal IN 0
USE depart IN 0
USE post IN 0
SELECT personal.*, depart.name as depart, post.name as post ;
   FROM personal ;
   LEFT JOIN depart ON depart.id_dep = personal.id_dep ;
   LEFT JOIN post ON post.id_post = personal.id_post ;
   ORDER BY personal.name INTO CURSOR cur_report
loExcel = CREATEOBJECT("Excel.Application")
WITH loExcel
   .Visible = .t.
   .WorkBooks.Add
*
* Устанавливаем ширину столбцов
*
    .Columns[1].ColumnWidth = 22        && Фамилия
    .Columns[2].ColumnWidth = 10        && Дата приема на работу
    .Columns[3].ColumnWidth = 8         && Телефон
    .Columns[4].ColumnWidth = 18        && Подразделение
    .Columns[5].ColumnWidth = 20        && Должность
*
* Выводим строку заголовка
*
   .Range("A1:E1").Select               && Выделяем область заголовка
   WITH .Selection
      .MergeCells = .t.                 && Объединяем ячейки
      .HorizontalAlignment = xlCenter   && Текст в ячейках центрируется
      .VerticalAlignment = xlCenter     && по горизонтали и вертикали
   ENDWITH
   WITH .ActiveCell                     && В образованную ячейку
      .Value = "Список сотрудников"     && записываем текст заголовка
      .Font.Size = 14                   && и устанавливаем
      .Font.Bold = .t.                  && параметры шрифта
   ENDWITH
*
* Форматируем строку для шапки таблицы
*
   .Rows("2:2").RowHeight = 18          && Устанавливаем ширину строки 2
   .Range("A2:E2").Select               && Выделяем область шапки
   WITH .Selection
```

```
      .HorizontalAlignment = xlCenter   && Текст в ячейках центрируется
      .VerticalAlignment = xlCenter     && по горизонтали и вертикали
      .Interior.Pattern = xlSolid       && Определяем сплошную кисть
      .Interior.Color = RGB(255,255,224)&& и заливаем заголовок желтым
   ENDWITH                              && цветом
*
* Выводим наименования колонок
*
   .Cells(2,1).value = "Фамилия и имя"  && Ячейки с 1 по 5 на 2-й строке
   .Cells(2,2).value = "Принят"
   .Cells(2,3).value = "Телефон"
   .Cells(2,4).value = "Подразделение"
   .Cells(2,5).value = "Должность"
*
* Заполняем таблицу данными из курсора cur_report
*
   lnRow = 2                              && Начальный номер строки
   SELECT cur_report
   SCAN
      lnRow = lnRow + 1                 && Модифицируем счетчик строк
      .Cells(lnRow,1).value = ALLTRIM(cur_report.name)
      .Cells(lnRow,2).value = cur_report.datbegin
      .Cells(lnRow,3).value = ALLTRIM(cur_report.tlf)
      .Cells(lnRow,4).value = ALLTRIM(cur_report.depart)
      .Cells(lnRow,5).value = ALLTRIM(cur_report.post)
   ENDSCAN
*
* Обрамляем таблицу рамкой
*
   .Range(.Cells(2,1), .Cells(lnRow,5)).Select  && Область таблицы,
   WITH .Selection                              && включая заголовок
      .Borders(xlEdgeLeft).Weight = xlMedium    && Обрамление краев
      .Borders(xlEdgeTop).Weight = xlMedium     && линиями средней
      .Borders(xlEdgeBottom).Weight = xlMedium  && толщины
      .Borders(xlEdgeRight).Weight = xlMedium
      .Borders(xlInsideVertical).Weight = xlThin   && Внутренние тонкие
      .Borders(xlInsideHorizontal).Weight = xlThin && линии
   ENDWITH
*
* Отделяем двойной чертой шапку таблицы от строк
*
   .Range("A2:E2").Select                 && Выделяем область ячеек
   WITH .Selection.Borders(xlEdgeBottom) && Для нижней линии ячеек
      .LineStyle = xlDouble               && Двойная толстая
      .Weight = xlThick                   && линия
   ENDWITH
ENDWITH
```

В коде оператором `SELECT` данные выбираются в курсор `cur_report` и затем выводятся на лист Microsoft Excel. Использованные приемы форматирования отчета прокомментированы в листинге.

Результат выполнения кода листинга 17.12 показан на рис. 17.19.

Формирование отчета в виде иерархической таблицы

Используя таблицы **Personal**, **Depart** и **Post**, построим иерархическую таблицу, в которой перегруппируем заголовки таким образом, чтобы первая колонка содержала

	A	B	C	D	E
1	**Список сотрудников**				
2	Фамилия и имя	Принят	Телефон	Подразделение	Должность
3	Александров Иван	02.03.2000	73	Технический	Техник
4	Александрова Ирина	04.03.2001	73	Технический	Техник
5	Иванов Александр	12.01.2003	91	Администрация	Главный инженер
6	Иванов Иван	01.10.2001	92	Администрация	Директор
7	Иванова Тамара	06.06.2002	90	Администрация	Секретарь
8	Петров Пётр	05.05.1999	71	Технический	Инженер
9	Посредников Пётр	18.08.1999	70	Технический	Начальник отдела
10	Прошкин Сергей	15.12.2004	76	Технический	Инженер
11	Семёнов Семён	12.01.2001	75	Технический	Инженер
12	Сидорова Вера	06.07.2001	72	Технический	Инженер
13	Страхова Вера	10.11.2003	74	Бухгалтерия	Главный бухгалтер
14	Тимошенко Юлия	15.05.2006	55	Бухгалтерия	Бухгалтер-кассир

Рис. 17.19. Отчет в виде плоской таблицы

список подразделений, вторая — список должностей, а оставшиеся колонки — фамилию и имя сотрудника, дату приема на работу и телефон. Особенность формирования иерархической таблицы заключается в том, что ячейки, содержащие одинаковые значения, объединяются, чем исключается дублирование наименований подразделений и должностей.

Код для формирования иерархической таблицы приведен в листинге 17.13.

```
#DEFINE xlTop            -4160
#DEFINE xlCenter         -4108
#DEFINE xlSolid              1
#DEFINE xlEdgeLeft           7
#DEFINE xlEdgeTop            8
#DEFINE xlEdgeBottom         9
#DEFINE xlEdgeRight         10
#DEFINE xlInsideVertical    11
#DEFINE xlInsideHorizontal  12
#DEFINE xlContinuous         1
```

```
#DEFINE xlThin               2
#DEFINE xlThick              4
#DEFINE xlMedium         -4138
#DEFINE xlAutomatic      -4105
#DEFINE xlDouble         -4119
LOCAL lcPath, loExcel, lnRow
lcPath = JUSTPATH(SYS(16))
SET DEFAULT TO (lcPath)
CLOSE TABLES ALL
USE personal IN 0
USE depart IN 0
USE post IN 0
loExcel = CREATEOBJECT("Excel.Application")
WITH loExcel
   .Visible = .t.
   .WorkBooks.Add
*
* Форматируем столбцы
*
    .Columns[1].ColumnWidth = 18   && Подразделение
    .Columns[2].ColumnWidth = 20   && Должность
    .Columns[3].ColumnWidth = 22   && Фамилия
    .Columns[4].ColumnWidth = 10   && Дата приема на работу
    .Columns[5].ColumnWidth = 8    && Телефон
*
* Выводим строку заголовка
*
   .Range("A1:E1").Select          && Выделяем область заголовка
   WITH .Selection
      .MergeCells = .t.            && Объединяем ячейки
      .HorizontalAlignment = xlCenter
      .VerticalAlignment = xlCenter
   ENDWITH
   WITH .ActiveCell
      .Value = "Список сотрудников"
      .Font.Size = 14
      .Font.Bold = .t.
   ENDWITH
*
* Форматируем строку для шапки таблицы
*
   .Rows("2:2").RowHeight = 18     && Высота строки шапки таблицы
   .Range("A2:E2").Select
   WITH .Selection
      .HorizontalAlignment = xlCenter
      .VerticalAlignment = xlCenter
```

```
      .Interior.Pattern = xlSolid        && Шапка таблицы заливается
      .Interior.Color = RGB(255,255,224) && кистью желтого цвета
   ENDWITH
*
* Выводим наименования колонок
*
   .Cells(2,1).value = "Подразделение"
   .Cells(2,2).value = "Должность"
   .Cells(2,3).value = "Фамилия и имя"
   .Cells(2,4).value = "Принят"
   .Cells(2,5).value = "Телефон"
*
* Заполняем таблицу данными из курсора temp
* Обратите внимание на то, как отсортированы данные в курсоре temp
* Такая сортировка необходима для правильного построения таблицы
*
   SELECT * FROM personal ORDER BY id_dep, id_post, name INTO CURSOR temp
*
* Курсор temp1 содержит количество записей в таблице Personal
* для каждого подразделения; сортировка по подразделениям
*
   SELECT id_dep, COUNT(id_dep) as cpos FROM temp GROUP BY id_dep ;
      ORDER BY id_dep INTO CURSOR temp1
   lnStartRow = 3
   SELECT temp1
   SCAN                               && ... Для первой колонки таблицы
      .Range(.Cells(lnStartRow,1), ;
             .Cells(lnStartRow + temp1.cpos - 1,1)).Select
      WITH .Selection                 && Определение области
         .MergeCells = .t.            && и объединение ячеек
         .VerticalAlignment = xlTop && Текст - по верхнему краю
      ENDWITH
      SEEK temp1.id_dep ORDER TAG id_dep IN depart
      .ActiveCell.value = ALLTRIM(depart.name)
      lnStartRow = lnStartRow + temp1.cpos
   ENDSCAN
   USE IN temp1
*
* Курсор temp1 содержит количество записей в таблице Personal
* для каждой должности; сортировка по должностям
*
   SELECT id_post, COUNT(id_post) as cpos FROM temp GROUP BY id_post ;
      ORDER BY id_post INTO CURSOR temp1
   lnStartRow = 3
   SELECT temp1
   SCAN                               && ... Для второй колонки таблицы
```

```
      .Range(.Cells(lnStartRow,2), ;
             .Cells(lnStartRow + temp1.cpos - 1,2)).Select
      WITH .Selection               && Определение области
         .MergeCells = .t.          && и объединение ячеек
         .VerticalAlignment = xlTop &&Текст - по верхнему краю
      ENDWITH
      SEEK temp1.id_post ORDER TAG id_post IN post
      .ActiveCell.value = ALLTRIM(post.name)
      lnStartRow = lnStartRow + temp1.cpos
   ENDSCAN
   USE IN temp1
   SELECT temp
   GO TOP
   lnRow = 2
   SCAN                     && Выводим поля из таблицы Personal
      lnRow = lnRow + 1
      .Cells(lnRow,3).value = ALLTRIM(temp.name)
      .Cells(lnRow,4).value = temp.datbegin
      .Cells(lnRow,5).value = ALLTRIM(temp.tlf)
   ENDSCAN
   USE IN temp
*
* Обрамляем таблицу рамкой
*
   .Range(.Cells(2,1), .Cells(lnRow,5)).Select
   WITH .Selection
      .Borders(xlEdgeLeft).Weight = xlMedium
      .Borders(xlEdgeTop).Weight = xlMedium
      .Borders(xlEdgeBottom).Weight = xlMedium
      .Borders(xlEdgeRight).Weight = xlMedium
      .Borders(xlInsideVertical).Weight = xlThin
      .Borders(xlInsideHorizontal).Weight = xlThin
   ENDWITH
*
* Отделяем двойной чертой заголовок от строк таблицы
*
   .Range("A2:E2").Select
   WITH .Selection.Borders(xlEdgeBottom)
      .LineStyle = xlDouble
      .Weight = xlThick
   ENDWITH
ENDWITH
```

Результат выполнения кода листинга 17.13 показан на рис. 17.20.

	A	B	C	D	E
1	**Список сотрудников**				
2	Подразделение	Должность	Фамилия и имя	Принят	Телефон
3	Администрация	Директор	Иванов Иван	01.10.2001	92
4		Главный инженер	Иванов Александр	12.01.2003	91
5		Секретарь	Иванова Тамара	06.06.2002	90
6	Бухгалтерия	Главный бухгалтер	Страхова Вера	10.11.2003	74
7		Бухгалтер-кассир	Тимошенко Юлия	15.05.2006	55
8	Технический	Начальник отдела	Посредников Пётр	18.08.1999	70
9		Инженер	Петров Пётр	05.05.1999	71
10			Прошкин Сергей	15.12.2004	76
11			Семёнов Семён	12.01.2001	75
12			Сидорова Вера	06.07.2001	72
13		Техник	Александров Иван	02.03.2000	73
14			Александрова Ирина	04.03.2001	73

Рис. 17.20. Иерархическая таблица

Уменьшение времени формирования отчета

Вы можете значительно ускорить формирование отчета, особенно при больших объемах выводимых данных, используя приведенные ниже рекомендации.

"Сокрытие" окна Microsoft Excel

Выполняйте все действия по выводу данных в режиме "скрытого" главного окна Microsoft Excel. Для этого устанавливайте свойство `Visible` объекта `Application` в "истину" только после того, как отчет будет сформирован — в этом случае не будет тратиться время на рисование документа на экране монитора. Практически во всех рассмотренных нами примерах окно Excel объявлялось видимым сразу после соединения с сервером автоматизации, но это делалось в учебных целях. Впрочем, возможно, пользователям будет интересно наблюдать за автоматическим формированием документа, если это не занимает несколько часов...

Использование объектных ссылок

Во всех предыдущих примерах мы постоянно создавали объектные ссылки для объектов Microsoft Excel. Например, для ссылки на объект `Application` применялась переменная `loExcel`, для ссылки на активную книгу — переменная `loBook`, и, наконец, для ссылки на активный лист — переменная `loSheet`.

Использование объектных ссылок позволяет уменьшить время доступа к объекту в иерархии. Вспомните, что автоматизация подразумевает использование позднего связывания через интерфейс IDispath. А т. к. каждый объект, как вы помните, предоставляет свой интерфейс, унаследованный от IDispath, то, чем сложнее объектная ссылка, тем больше времени требуется на ее раскрытие.

Например, если для установки свойства объекта используется команда вида

```
BaseObject.Object1.Object2.Property = Value
```

то первоначально из объекта *BaseObject* запрашивается интерфейс объекта *Object1* и вызываются его методы для получения свойства *Object2*. Так как это свойство является ссылкой на другой объект, то запрашивается интерфейс этого объекта и вызываются его методы для получения свойства *Property*, что и определяет увеличение времени доступа. Поэтому старайтесь по возможности создавать и использовать объектные ссылки. Но при этом вы должны помнить, что объект существует, пока существует хотя бы одна ссылка на него!

Использование псевдонимов

Язык VBA предоставляет специальные имена, или псевдонимы, позволяющие получать доступ к активному объекту непосредственно из объекта `Application`:

- ActiveWindow — активное окно;
- ActiveWorkBook — активная рабочая книга;
- ActiveWorkSheet — активный рабочий лист;
- ActiveChart — активная диаграмма;
- ActiveCell — активная ячейка.

Использование массивов для передачи данных

Еще один способ, который можно использовать для таблиц — это передача данных через массивы; этот способ позволяет повысить быстродействие в разы (и даже десятки раз). Для его реализации нужно выполнить запрос, сохраняющий результат выборки данных из таблицы в массиве, и затем передать указатель на этот массив в Microsoft Excel для заполнения выделенной области считанными данными (листинг 17.14).

```
*PROCEDURE ArrayToExcel
LOCAL lnRows, lnColumns, lcFile, loExcel
PUBLIC gaArray[1]
CLOSE TABLES ALL
lcFile = GETFILE("DBF")     && Вызов диалога для выбора таблицы
IF !EMPTY(lcFile)           && Если файл таблицы выбран
   USE (lcFile) IN 0 ALIAS MyTable
   SELECT * FROM MyTable INTO ARRAY gaArray && Выборка данных в массив
*
* Определение размерностей массива, которые будут использованы
* для определения области в Microsoft Excel
*
   lnRows = ALEN(gaArray, 1)    && Количество строк массива
   lnColumns = ALEN(gaArray, 2) && Количество столбцов
```

```
   loExcel = CREATEOBJECT("Excel.Application")
   WITH loExcel
      .WorkBooks.Add
      .Visible = .t.
      .Range(.Cells(1,1), ;
             .Cells(lnRows, lnColumns)).value = GetArrayPtr("gaArray")
   ENDWITH
   USE IN MyTable
ENDIF
RELEASE gaArray
*
* В функции GetArrayPtr используется недокументированный
* формат оператора RETURN, позволяющий возвратить
* указатель на массив
*
FUNCTION GetArrayPtr(taArrayName)
RETURN @&taArrayName
```

В коде используется недокументированная возможность Visual FoxPro возвращать указатель на массив при помощи оператора

```
RETURN @&cArrayName
```

где *cArrayName* — передаваемое функции имя массива. Массив с именем, определяемым *cArrayName*, должен быть "виден" в функции, возвращающей указатель на него, т. е. он должен быть объявлен как `PUBLIC` или `PRIVATE`.

Для сохранения возвращаемого функцией `GetArrayPtr` нельзя использовать какие-либо промежуточные переменные, как, например, в следующем фрагменте кода:

```
Ptr = GetArrayPtr("gaArray")
Object.Range(.Cells(1,1), .Cells(lnRows, lnColumns)).value = Ptr
```

т. к. это приведет к тому, что в переменной `Ptr` будет сохранено значение первого элемента массива, которое и будет присвоено всем ячейкам области.

Печать

Перед печатью документа нужно настроить параметры страницы (размеры и ориентация листа бумаги), а также, возможно, определить такие элементы, как колонтитулы или строку (группу строк), которые будут повторяться на каждом очередном листе отчета (например, шапка таблицы). И, наконец, вы должны указать, что именно должно быть напечатано: вся рабочая книга, только рабочий лист или диаграмма.

Объект *PageSetup*: настройка параметров листа принтера

Вы можете установить формат и ориентацию страницы отчета и величину отступов от границ листа, а также определить многие другие параметры форматирования, используя объект `PageSetup`. Этот объект доступен из объектов `WorkSheet` и `Chart`, что

позволяет использовать различное форматирование страниц отчета для разных рабочих листов и диаграмм.

Объект `PageSetup` предоставляет более двух десятков свойств для форматирования страницы отчета; мы рассмотрим здесь наиболее часто используемые свойства.

Свойство `PaperSize` позволяет определить формат листа:

```
oSheet.PageSetup.PaperSize = Constants
```

Свойство `Orientation` определяет ориентацию листа:

```
oSheet.PageSetup.Orientation = Constants
```

Здесь *`Constants`* — значение предопределенной константы Microsoft Excel.

Константы, определяющие формат и ориентацию листа, перечислены в табл. 17.9.

Свойство `CenterHorizontally` центрирует печатаемый документ по горизонтали, а свойство `CenterVertically` — по вертикали. Для центрирования установите значение свойства в "истину".

Свойства `LeftMargin`, `RightMargin`, `TopMargin` и `BottomMargin` определяют отступы от краев листа. Размер отступа устанавливается в пунктах (1/72 дюйма).

Таблица 17.9. *Константы, определяющие формат листа*

Константа	Значение	Описание
`xlPaperA5`	11	Лист размером 148×10 мм
`xlPaperA4`	9	Лист размером 210×297 мм
`xlPaperA3`	8	Лист размером 297×420 мм
`xlPortrait`	1	Книжная ориентация листа
`xlLandscape`	2	Альбомная ориентация листа

Вы можете вывести на печатаемый лист дополнительную информацию, используя следующие свойства:

- `LeftHeader` — вывод верхнего колонтитула, выровненного по правому краю листа;
- `CenterHeader` — вывод верхнего колонтитула, выровненного по центру листа;
- `RightHeader` — вывод верхнего колонтитула, выровненного по правому краю листа;
- `LeftFooter` — вывод нижнего колонтитула, выровненного по правому краю листа;
- `CenterFooter` — вывод нижнего колонтитула, выровненного по центру листа;
- `RightFooter` — вывод нижнего колонтитула, выровненного по правому краю листа.

Значения, которые вы можете присвоить этим свойствам, перечислены в табл. 17.10.

Таблица 17.10. *Коды значений, применяемые при выводе колонтитулов*

Код	Описание
&D	Печатает текущую дату
&T	Печатает текущее время
&F	Печатает наименование документа
&A	Печатает наименование рабочей книги
&P	Печатает номер страницы

Отступы для верхнего колонтитула определяются свойством `HeaderMargin`, а для нижнего — свойством `FooterMargin`.

В листинге 17.15 показан код для настройки параметров печати. Предполагается, что переменная `oSheet` содержит ссылку на рабочий лист.

```
#DEFINE xlLandscape  2          && Объявления констант
#DEFINE xlPaperA4    9
WITH oSheet.PageSetup
   .Orientation = xlLandscape  && Альбомная ориентация документа
   .PaperSize = xlPaperA4      && Формат A4
   .LeftMargin = 58            && Отступ слева — 2 см
   .RightMargin = 58           && Отступ справа — 2 см
   .TopMargin = 42             && Отступ сверху — 1,5 см
   .BottomMargin = 42          && Отступ снизу — 1,5 см
   .HeaderMargin = 29          && Отступ верхнего колонтитула — 1 см
   .FooterMargin = 29          && Отступ нижнего колонтитула — 1 см
   .LeftHeader = "&D &T"       && Слева вверху — дата и время
   .RightHeader = "&F"         && Справа вверху — наименование документа
   .CenterFooter = "Страница № &P"  && По центру внизу — номер страницы
ENDWITH
```

Вы можете определить группу строк рабочего листа как заголовок, который будет выводиться на каждой странице отчета. Например, если шапка таблицы в отчете занимает вторую и третью строки рабочего листа (в первой строке, вероятнее всего, располагается заголовок документа), то, указав номера этих двух строк в свойстве `PrintTitleRows` объекта `PageSetup`, вы заставите Microsoft Excel рисовать эту шапку на каждой странице отчета:

```
oSheet.PageSetup.PrintTitleRows = oSheet.Rows("2:3").Address
```

Свойство `PrintTitleColumns` по функциональности похоже на свойство `PrintTitleRows`; оно позволяет дублировать на каждом листе отчета расположенные слева колонки в

случае, если ваш отчет по ширине не помещается на лист бумаги. В следующем фрагменте кода на каждом печатном листе будет повторяться содержимое самой левой колонки:

```
oSheet.PageSetup.PrintTitleColumns = oSheet.Columns(1).Address
```

Метод *PrintOut*: вывод отчета на принтер

Для печати отчетов используется метод `PrintOut`, доступный из объектов `WorkBook` (рабочая книга), `WorkSheet` (рабочий лист) и `Chart` (диаграмма). Этот метод имеет несколько необязательных параметров, с которыми вы можете ознакомиться в справочной документации по VBA. При вызове метода без параметров печать будет выполнена на принтере, используемом по умолчанию:

```
Object.PrintOut
```

где *Object* — ссылка на объект, экспонирующий метод `PrintOut`.

Если вы вызываете этот метод из объекта `WorkBook`, то будет напечатана вся книга. Для печати конкретного рабочего листа используйте следующий код:

```
Object.Sheets(имя или индекс листа).PrintOut && Печать любого листа
```

или

```
Object.ActiveSheet.PrintOut          && Печать активного рабочего листа
```

где *Object* — ссылка на объект `Application` или `WorkBook`.

Метод `PrintOut` можно использовать для печати на принтере выделенной области рабочего листа:

```
oSheet.Range("A2:E2").PrintOut
```

Вы можете напечатать несколько копий отчетов, указав параметр `Copies` метода `PrintOut`. Этот параметр — третий в списке параметров метода.

```
Object.ActiveSheet.PrintOut(,,3)    && Печатать три копии рабочего листа
```

Уничтожение объекта Microsoft Excel

Способ уничтожения COM-объекта, созданного на базе Microsoft Excel, зависит от того, сделали вы его главное окно видимым или нет.

Если главное окно Excel не видимо (т. е. вы не устанавливали свойство `Visible` объекта `Application` в "истину"), то объект будет уничтожен, когда переменная, содержащая ссылку на него, перестанет существовать или ей будет присвоено иное значение. Если вы создали несколько объектных ссылок, до объект будет существовать, пока существует хотя бы одна ссылка на него.

Но если главное окно Excel отображается, то время жизни ссылочной переменной никак не влияет на время жизни связанного с ней объекта — этой особенностью мы пользовались во всех рассмотренных примерах, объявляя ссылочные переменные как

локальные. Уничтожить такой объект вы можете, только явно вызвав его метод `Quit()`:

```
oExcel.Quit()
```

При уничтожении объекта (или при закрытии рабочей книги) Microsoft Excel проверяет, были ли сохранены сделанные изменения. Если изменения не сохранены, то будет выведено диалоговое окно с запросом на сохранение документа. Для подавления этого диалога установите в "ложь" свойство `DisplayAlerts` объекта `Application`:

```
oExcel.DisplayAlerts = .f.
```

Автоматизация Microsoft Word

В предыдущих разделах этой главы вы узнали, как можно формировать отчеты в Microsoft Excel. Возможностей, предоставляемых этим компонентом Microsoft Office, обычно бывает вполне достаточно. Но не исключено, что у вас может возникнуть необходимость формировать отчеты в формате документов Microsoft Word.

Мы не будем рассматривать здесь все возможности, предоставляемые этим мощнейшим текстовым процессором. Основное внимание будет уделено вопросам программного вывода текста в документ, его структурированию и форматированию. Кроме того, будут рассмотрены вопросы, связанные с совместным использованием Microsoft Word и Microsoft Excel, а также печать документа и его отправка по факсу или электронной почте.

Объектная модель Microsoft Word

В вершине иерархии объектов Microsoft Word находится объект `Application`. Он включает коллекцию **Documents**, содержащую объекты `Document` для каждого открытого документа.

Имея в наличии документ, можно как угодно манипулировать с его содержанием. Можно добавлять, удалять и перемещать текст, изменять форматирование, добавлять сноски, рисунки, колонтитулы и т. д. Все действия над документом выполняются объектами, иерархически подчиненными объекту `Document`.

Объект *Document* и коллекция *Documents*

Вы можете создать новый или открыть уже существующий документ. Как только работа с документом завершена, он должен быть сохранен (по крайней мере, в большинстве случаев) и закрыт.

Для создания нового документа используется метод `Add` коллекции **Documents**, который возвращает ссылку на объект `Document`:

```
oWord = CREATEOBJECT("Word.Application")
oWordDoc = oWord.Documents.Add
```

Создать новый документ и сразу присвоить ему имя можно следующим образом:

```
oWordDoc = oWord.Documents.Add.SaveAs(cFileName)
```

где *cFileName* — путь и имя файла документа.

При создании документа вы можете задать шаблон, на котором будет основываться новый документ:

```
oWordDoc = oWord.Documents.Add(cTemplate, lNewTemplate)
```

где *cTemplate* указывает путь и имя файла шаблона (с расширением dot), а *lNewTemplate* определяет, будет ли создаваемый документ шаблоном или обычным документом. По умолчанию значение *lNewTemplate* — "ложь" (обычный документ).

Для открытия существующего документа используется метод Open:

```
oWord = CREATEOBJECT("Word.Application")
oWordDoc = oWord.Documents.Open(cFileName)
```

где *cFileName* — путь и имя файла документа.

Метод Close коллекции **Documents** используется для закрытия документа:

```
oWordDoc.Close(SaveChange, OriginalFormat, RouteDocument)
```

Параметр *SaveChange* определяет, нужно ли сохранять документ в файле. Его возможные значения показаны в табл. 17.11.

Параметр *OriginalFormat* задает формат, в котором должен быть сохранен документ. Его возможные значения показаны в табл. 17.12.

Параметр *RouteDocument* имеет логическое значение; он определяет, должен ли документ отсылаться следующему получателю в списке рассылки.

Таблица 17.11. *Значения параметра SaveChange метода Close объекта документа*

Константа Word	Значение	Описание
wdDoNotSaveDocument	0	Документ не сохраняется
wdPromptToSaveChanges	–2	При закрытии документа выводится диалог с запросом о сохранении
wdSaveChanges	–1	Изменения сохраняются (если имя файла документа не известно, то будет выведен диалог с запросом на сохранение файла)

Таблица 17.12. *Значения параметра SaveChange метода Close объекта документа*

Константа Word	Значение	Описание
wdOriginalDocumentFormat	1	Документ сохраняется в исходном формате
wdPromptUser	2	Формат документа запрашивается у пользователя

wdWordDocument	0	Документ сохраняется в формате Word

Объект *Selection*

Многое из того, что делается с содержимым документа, предполагает наличие некоторой заданной позиции или части (области) документа, над которой нужно произвести некоторые действия. Для ее определения используются объекты `Selection` и `Range`.

Объект `Selection` предоставляет выделенный текст в документе. Этот объект существует всегда, даже когда в документе нет никакого текста; доступ к нему можно получить из объекта `Application`.

> ***ЗАМЕЧАНИЕ***
>
> К объекту `Selection` можно получить доступ также из объектов `Window` и `Pane`. Так как эти объекты ориентированы на поддержку интерактивной работы с пользователем, то здесь они рассматриваться не будут.

Выделяемая объектом `Selection` область может быть пустой, т. е. не содержать ни одного символа; при вставке в документ каждого очередного символа выделенная область увеличивается, включая в себя этот символ.

Мы рассмотрим здесь только те свойства и методы объекта `Selection`, которые могут быть использованы для формирования отчетов.

Метод `TypeText` добавляет переданный ему текст в документ:

```
oWord.Selection.TypeText(cText)
```

Строка *`cText`* может содержать любые символы; вы можете использовать функцию `CHR()` для включения в строку служебных символов. Включение в строку последовательности символов возврата каретки и перевода строки формирует абзац.

Метод `TypeParagraph` вставляет в текст документа признак окончания абзаца и закрывает выделенную область. Метод не имеет параметров:

```
oWord.Selection.TypeParagraph
```

Все действия по форматированию выполняются над выделенной областью.

Объект *Font*

Объект `Font` предназначен для установки параметров используемого шрифта. На него можно ссылаться из множества других объектов иерархии Microsoft Word; объект `Selection` также имеет свойство `Font`, ссылающееся на объект `Font`. В листинге 17.16 показан пример создания нового документа, в который записывается несколько разнородных строк.

```
LOCAL loWord
loWord = CREATEOBJECT("Word.Application")
loWord.Visible = .t.
loWord.Documents.Add
WITH loWord.Selection
   .TypeText("Привет! Текст, который вы видите перед собой, " + ;
             "создан программно. ")
   .Font.Bold = .t.
   .TypeText("Сейчас это жирный шрифт, ")
   .Font.Italic = .t.
   .TypeText("а теперь — курсив." + CHR(10) + CHR(13))
   .Font.Bold = .t.
   .Font.Italic = .f.
   .Font.Size = 20
   .Font.Color = RGB(0, 128, 0)
   .TypeText("It is big green string!")
   .TypeParagraph
ENDWITH
```

В примере использованы стандартные свойства объекта `Font` для управления толщиной символов, курсивом и цветом (см. табл. 17.3). Вы можете изменять цвет символов, применяя свойство `ColorIndex`, которое для определения цвета использует индексы. При использовании свойства `Color` помните, что значения цветовых компонентов, передаваемых функции `RGB()`, должны быть кратными 32.

Для изменения шрифта используется свойство `Name`:

```
oWord.Selection.Font.Name = "Arial Cyr"
```

Объект *ParagraphFormat*

Как следует из названия, объект `ParagraphFormat` отвечает за форматирование абзаца или группы абзацев. Объект `Selection` имеет свойство `ParagraphFormat`, которое ссылается на одноименный объект. Рассмотрим его основные свойства.

Свойство `Alignment` отвечает за выравнивание текста и может принимать одно из предопределенных значений, определяемых показанными в табл. 17.13 константами Microsoft Word.

Таблица 17.13. *Константы Microsoft Word для форматирования абзацев*

Константа Word	Значение	Описание
wdAlignParagraphLeft	0	Выравнивание текста по левому краю
wdAlignParagraphCenter	1	Центрирование текста
wdAlignParagraphRight	2	Выравнивание текста по правому краю
wdAlignParagraphJustify	3	Выравнивание текста по ширине

Свойство `Style` определяет, что именно заключено в абзаце. Из всех возможных стилей мы рассмотрим только те, которые позволяют определить нормальный текст или заголовок. Константы, определяющие стили и их значения, перечислены в табл. 17.14.

Таблица 17.14. *Константы Microsoft Word, определяющие стили*

Константа Word	Значение	Описание
`wdStyleNormal`	–1	Обычный текст
`wdStyleHeading1`	–2	Заголовок 1
`wdStyleHeading2`	–3	Заголовок 2
`wdStyleHeading3`	–4	Заголовок 3
`wdStyleHeading4`	–5	Заголовок 4
`wdStyleHeading5`	–6	Заголовок 5
`wdStyleHeading6`	–7	Заголовок 6
`wdStyleHeading7`	–8	Заголовок 7

Рекомендуем вам использовать перечисленные в таблице заранее предопределенные заголовки. В этом случае вы сможете автоматически формировать содержание документа.

Изменим код листинга 17.16: "большую зеленую строку" выведем по центру листа, а также добавим несколько заголовков для демонстрации использования стилей (листинг 17.17).

```
#DEFINE wdStyleNormal            -1
#DEFINE wdStyleHeading1        -2
#DEFINE wdStyleHeading2        -3
#DEFINE wdStyleHeading3        -4
#DEFINE wdStyleHeading4        -5
#DEFINE wdStyleHeading5        -6
#DEFINE wdStyleHeading6        -7
#DEFINE wdAlignParagraphCenter  1
LOCAL loWord
loWord = CREATEOBJECT("Word.Application")
loWord.Visible = .t.
loWord.Documents.Add
WITH loWord.Selection
   .TypeText("Привет! Текст, который вы видите перед собой, " + ;
```

```
            "создан программно. ")
   .Font.Bold = .t.
   .TypeText("Сейчас это жирный шрифт, ")
   .Font.Italic = .t.
   .TypeText("а теперь — курсив." + CHR(10) + CHR(13))
   .TypeParagraph
   .Font.Bold = .t.
   .Font.Italic = .f.
   .Font.Size = 20
   .Font.Color = RGB(0, 128, 0)
   .TypeText("It is big green string!")
   .ParagraphFormat.Alignment = wdAlignParagraphCenter
   .TypeParagraph
   .TypeText("Заголовок 1")
   .ParagraphFormat.Style = wdStyleHeading1
   .TypeParagraph
   .TypeText("Заголовок 2")
   .ParagraphFormat.Style = wdStyleHeading2
   .TypeParagraph
   .TypeText("Заголовок 3")
   .ParagraphFormat.Style = wdStyleHeading3
   .TypeParagraph
   .TypeText("Заголовок 4")
   .ParagraphFormat.Style = wdStyleHeading4
   .TypeParagraph
   .TypeText("Заголовок 5")
   .ParagraphFormat.Style = wdStyleHeading5
   .TypeParagraph
   .TypeText("Заголовок 6")
   .ParagraphFormat.Style = wdStyleHeading6
   .TypeParagraph
ENDWITH
```

Результат выполнения этого кода показан на рис. 17.21.

Свойство `LineSpacingRule` определяет расстояние между строками внутри абзаца. Допустимые значения этого свойства перечислены в табл. 17.15.

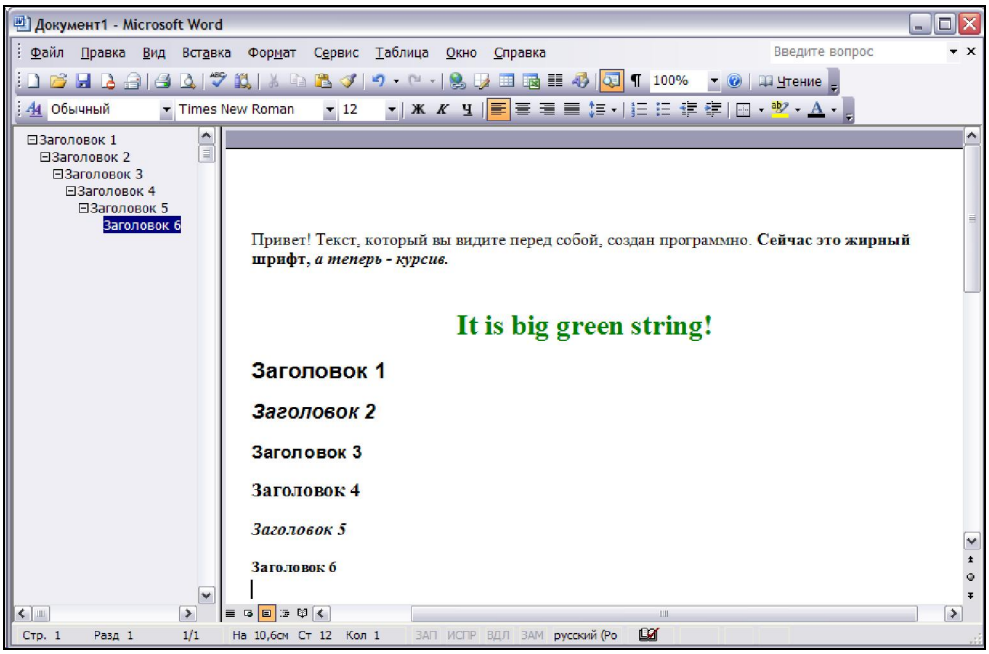


Рис. 17.21. Результат выполнения кода листинга 17.18

Таблица 17.15. *Константы Microsoft Word, определяющие междустрочное расстояние*

Константа Word	Значение	Описание
wdLineSpaceSingle	0	Обычный интервал
wdLineSpace1pt5	1	Полуторный интервал
wdLineSpaceDouble	2	Двойной интервал

Свойство `FirstLineIndent` определяет отступ (положительное значение) или выступ (отрицательное значение) для первой строки абзаца (в пунктах):

```
oWord.Selection.ParagraphFormat.FirstLineIndent = 42     && отступ 1,5 см
```

Свойства `LeftIndent` и `RightIndent` определяют отступ строк абзаца от левого и правого краев листа. Значения отступа также указываются в пунктах (1/72 дюйма).

В листинге 17.18 показан код, в котором выполняется форматирование текста, а на рис. 17.22 — результат выполнения этого кода.

```
#DEFINE wdAlignParagraphLeft     0
#DEFINE wdAlignParagraphCenter   1
```

```
#DEFINE wdAlignParagraphJustify  3
#DEFINE wdStyleNormal           -1
#DEFINE wdStyleHeading2         -3
LOCAL loWord
loWord = CREATEOBJECT("Word.Application")   && Создаем объект
loWord.Documents.Add                        && Новый документ
loWord.Visible = .t.                        && Сделать видимым
WITH loWord.Selection
   .ParagraphFormat.Style = wdStyleHeading2 && Заголовок 2 по центру
   .ParagraphFormat.Alignment = wdAlignParagraphCenter
   .TypeText("Объект Selection")            && Вывод заголовка
   .TypeParagraph                           && Конец абзаца
   .ParagraphFormat.Alignment = wdAlignParagraphJustify
   .ParagraphFormat.Style = wdStyleNormal
   .Font.Size = 10
   .TypeText("Объект Selection предоставляет выделенный текст в " + ;
       "документе. Этот объект существует всегда, даже когда в " + ;
       "документе  нет никакого текста; доступ к нему можно " + ;
       "получить из объекта Application.")
   .ParagraphFormat.FirstLineIndent = 20  && 1-я строка отступ 20 пунктов
   .TypeParagraph                         && Конец абзаца
   .TypeParagraph
   WITH .Font
      .Size = 10
      .Bold = .t.
      .Italic = .t.
   ENDWITH
   .ParagraphFormat.FirstLineIndent = 0   && Вывод "Замечания"
   .ParagraphFormat.LeftIndent = 30       && Отступ от левого и правого
   .ParagraphFormat.RightIndent = 30      && краев — 30 пунктов
   .ParagraphFormat.Alignment = wdAlignParagraphJustify
   .TypeText("Замечание")
   .TypeParagraph
   WITH .Font
      .Size = 9
      .Bold = .f.
      .Italic = .f.
   ENDWITH
   .TypeText("К объекту Selection можно получить доступ также из " + ;
       "объектов Window и Pane. Так как эти объекты ориентированы " + ;
       "на  поддержку интерактивной работы с пользователем, то  " + ;
       "здесь они рассматриваться не будут.")
   .TypeParagraph
   .TypeParagraph
   .ParagraphFormat.FirstLineIndent = 20 && 1-я строка отступ 20 пунктов
   .ParagraphFormat.LeftIndent = 0       && Отменяем отступ от левого
   .ParagraphFormat.RightIndent = 0      && и правого краев
```

```
    .ParagraphFormat.Alignment = wdAlignParagraphJustify
    .Font.Size = 10
    .TypeText("Выделяемая объектом Selection область может быть " + ;
        "пустой, т. е. не содержать ни одного символа; при вставке " + ;
        "в документ каждого очередного символа выделенная область " + ;
        "увеличивается, включая в себя этот символ.")
    .TypeParagraph
    .TypeText("Мы рассмотрим здесь только те свойства и методы " + ;
        "объекта Selection, которые могут быть использованы для " + ;
        "формирования отчетов.")
    .TypeParagraph
ENDWITH
```

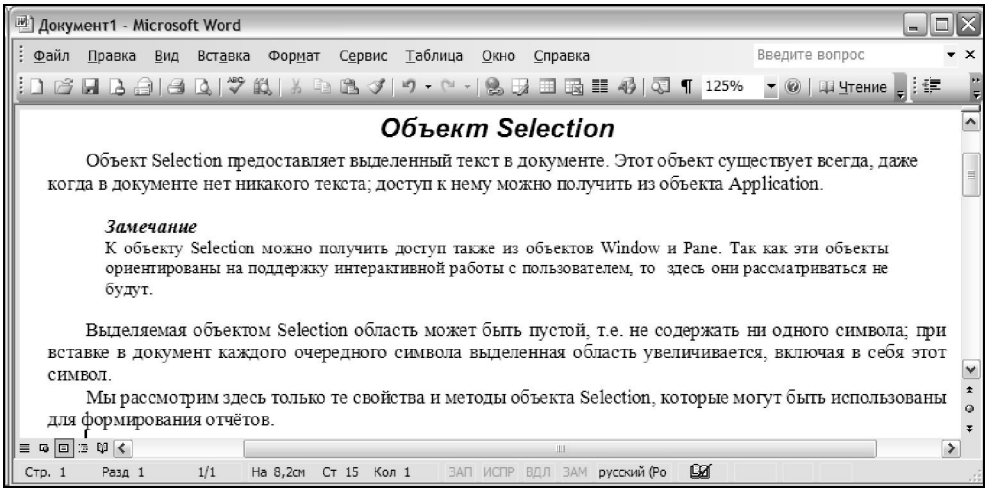


Рис. 17.22. Форматирование абзацев

Объект *Range*

Объект `Range` предоставляет раздел документа. Он определяется первым и последним символами текстового блока и может иметь размер от пустой строки до размера всего документа. Это самый "востребованный" в Microsoft Word объект; на рис. 17.23 показано окно Обозревателя объектов Visual FoxPro, в правой части которого перечислены интерфейсы объектов, экспонирующих объектную ссылку на объект `Range`.

Так как наша задача — понять, как правильно формировать отчеты в Microsoft Word, то использование этого объекта интересует нас прежде всего в контексте обработки закладок и формирования таблиц.

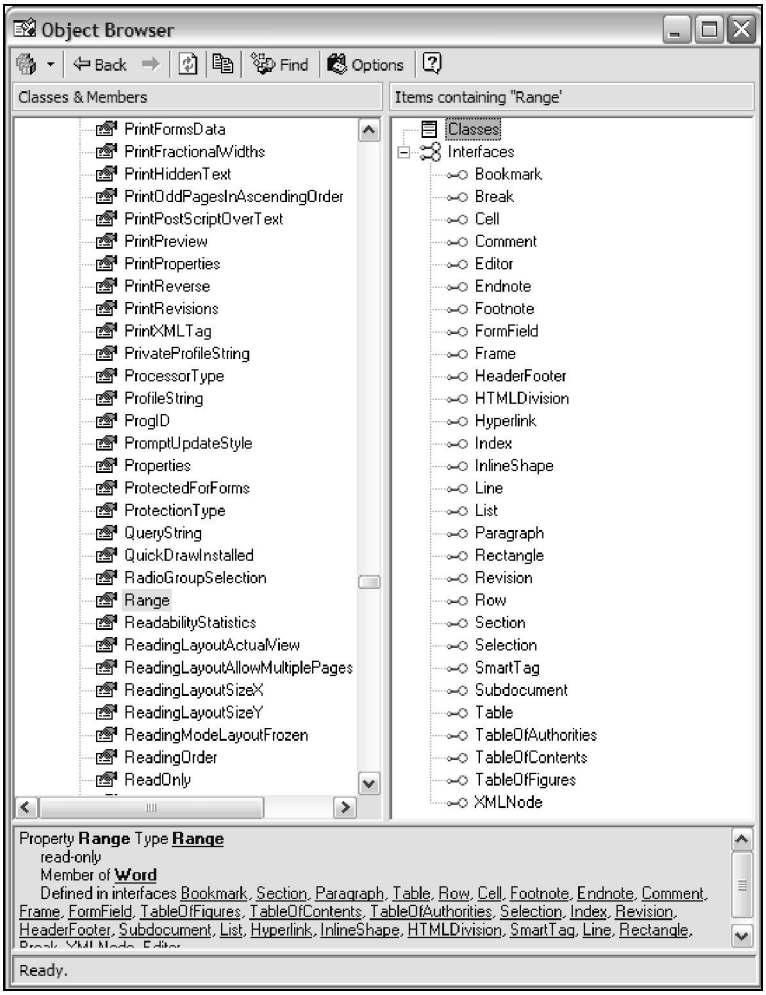


Рис. 17.23. Интерфейсы, экспонирующие ссылку на объект `Range`

Использование закладок

Закладка (BookMark) — это часть документа, которой присвоено имя. Закладка может не содержать никакого текста, и в этом случае она определяет лишь место внутри документа. Совокупность закладок образует коллекцию **BookMarks** документа.

Наиболее часто закладки используются для маркировки местоположения в документах и шаблонах документов. При открытии документа или создании нового документа на основании шаблона вы можете использовать закладки для вставки данных в документ. В листинге 17.19 показан код, в котором создается новый документ на основа-

нии шаблона **Response**, в котором в позиции закладок **Recipient** и **Product** вставляется информация из одноименных полей таблицы **Orders**.

```
loWord = CREATEOBJECT("Word.Application")
WITH loWord
   loDoc = .Documents.Add("Response.dot")
   loRange = loDoc.BookMarks("Recipient").Range
   loRange.InsertAfter(orders.recipient) && Текст в закладку Recipient
   loRange = loDoc.BookMarks("Product").Range
   loRange.InsertAfter(orders.product)   && Текст в закладку Product
   loWord.Visible = .t.
ENDWITH
```

Как видно из кода примера, коллекция **BookMarks** документа по имени закладки возвращает ссылку на объект `Range`; используя метод `InsertAfter` этого объекта, мы вставляем текст из поля таблицы непосредственно за закладкой.

Для того чтобы определить, существует ли некоторая закладка в документе, используется метод `Exists` коллекции **BookMarks**:

```
IF oDoc.BookMarks.Exists("SomeBookmarks")
* Код, обрабатывающий закладку
ELSE
* Сообщение об ошибке или иные действия из-за отсутствия закладки
ENDIF
```

где *oDoc* — объектная ссылка на объект `Document`.

Заключение

Мы надеемся, что по прочтении этой главы у вас не возникнет проблем в использовании приложений Microsoft Office для формирования отчетов из ваших приложений. За рамки этой главы вышло рассмотрение вопросов автоматизации других компонентов Microsoft Office; но, используя полученные знания, вы без труда освоите и Microsoft Outlook, и Microsoft PowerPoint. Мы также познакомились с далеко не всеми возможностями, которые могут быть реализованы в Microsoft Excel и Microsoft Word; в частности, в Microsoft Word можно также формировать отчеты, содержащие таблицы. Полагаем, что на основе изложенного здесь материала вы сможете самостоятельно освоить не рассмотренные здесь возможности Microsoft Office и эффективно применять их в ваших приложениях.