САМОУЧИТЕЛЬ

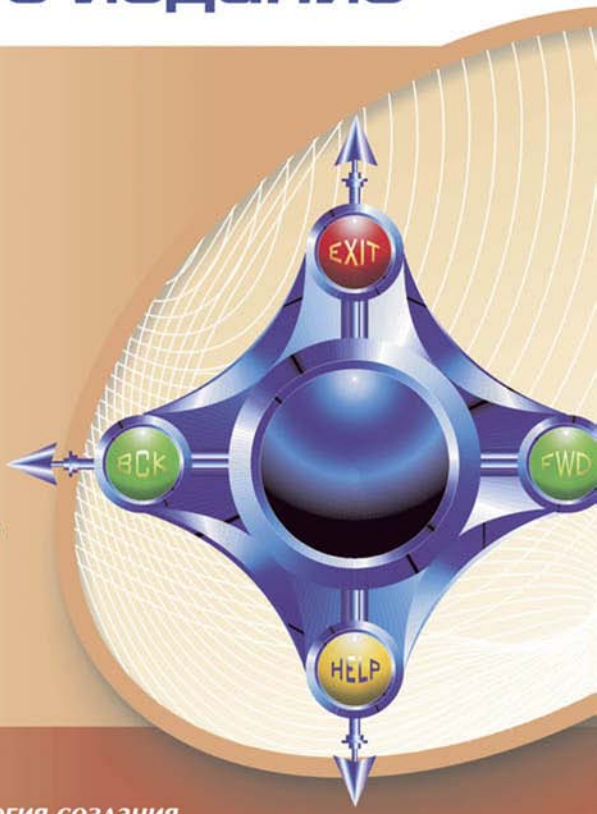
bhv

VBA

2-е издание

Андрей Гарнаев

- Автоматизация повседневной работы
- Около 350 примеров разработки приложений
- Основные элементы, объекты и методы
- Использование VBA для взаимодействия с Web и при работе с БД



Технология создания пользовательских приложений на примерах

Андрей Гарнаев

САМОУЧИТЕЛЬ

VBA

2-е издание

Санкт-Петербург

«БХВ-Петербург»

2004

УДК 681.3.068+800.92VBA
ББК 32.973.26-018.1
Г20

Гарнаев А. Ю.

Г20 Самоучитель VBA. — 2-е изд., перераб. и доп. — СПб.: БХВ-Петербург, 2004. — 560 с.: ил.

ISBN 978-5-94157-410-0

В книге изложены базовые сведения о VBA, работе с макросами, технологии ООП, конструировании пользовательского интерфейса и форм, работе с файлами, создании Web-служб и защите приложений. Приведены практические приемы автоматизации работы в офисных приложениях Excel, Word, Access и PowerPoint, обмена XML-документами между приложениями, а также управления работой разных приложений из одного файла, написанного на VBA. Показаны возможности интегрирования офисных приложений в .NET-приложения с использованием Microsoft Visual Studio Tools для Microsoft Office System. Рассмотрены вопросы создания скриптов для Windows, использования компонентов COM и Win32 API, программирования среды разработки кода и др. Книга насыщена большим числом примеров (около 350). Во втором, существенно переработанном и дополненном издании, отражены изменения последних лет в области разработки офисных приложений.

Для программистов

УДК 681.3.068+800.92VBA
ББК 32.973.26-018.1

Группа подготовки издания:

Главный редактор	*Екатерина Кондукова*
Зам. гл. редактора	*Евгений Рыбаков*
Зав. редакцией	*Григорий Добин*
Редактор	*Наталья Бубнова*
Компьютерная верстка	*Натальи Караваевой*
Корректор	*Светлана Симуни*
Дизайн обложки	*Игоря Цырульникова*
Зав. производством	*Николай Тверских*

Лицензия ИД № 02429 от 24.07.00. Подписано в печать 09.06.04.
Формат $70\times100^{1}/_{16}$. Печать офсетная. Усл. печ. л. 45,15.
Тираж 6000 экз. Заказ №
"БХВ-Петербург", 190005, Санкт-Петербург, Измайловский пр., 29.

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

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

ISBN 978-5-94157-410-0

Содержание

Предисловие

О чем данная книга

Операционная система Windows (особенно Windows XP и выше) корпорации Microsoft обладает изумительно удобным интерфейсом и множеством интеллектуальных средств, которые освобождают пользователя от рутинной работы, присущей другим операционным системам. Естественным и незаменимым дополнением этой операционной системы являются программные продукты Microsoft Office, которые даже начинающему пользователю позволяют с легкостью создавать идеальную по своему оформлению документацию (например, эта книга была написана и сверстана с помощью Microsoft Word), производить финансовые, инженерные и прочие расчеты при помощи Microsoft Excel, создавать и работать с базами данных средствами Microsoft Access, конструировать презентации и слайды с помощью Microsoft PowerPoint, Microsoft Outlook, создавать собственные Web-сайты средствами Microsoft FrontPage. Вместе, операционная система Windows и программные продукты Microsoft Office, покорили весь мир. Сейчас буквально нет ни одного офиса, ни одного дома, где бы не было компьютера с этими продуктами. Но это еще не все. Корпорация Microsoft интегрировала в свои офисные продукты, в саму операционную среду изумительный по простоте и необычайный по своей эффективности язык программирования Visual Basic for Applications или сокращенно VBA. С помощью этого языка теперь каждый пользователь может автоматизировать работу приложения и максимально приспособить его работу для решения текущих задач, не только добавив интерфейсу новую функциональность, но и удалив из него ненужные для данного приложения элементы, тем самым обеспечив ему дополнительную стабильность, своеобразную защиту от дурака. Например, если вам поручено в фирме ежедневно по результатам представлять сводную таблицу и диаграмму ее работы, наверное, уже через пару дней вам придет идея, почему бы эту процедуру не автоматизировать, т. к. программный инструментарий используется один и тот же, только источники данных разные. Здесь как раз вам на помощь придет VBA. А результат от созданного продукта будет двоякий — во-первых,

вы освободитесь от ежедневной работы, а во-вторых, получите шанс, чтобы босс вас заметил и понял, что вы действительно незаменимый сотрудник. VBA позволяет не только автоматизировать работу одного приложения, но и интегрировать работу нескольких в единое целое. Например, в нашем примере данные по работе фирмы пусть хранятся в базе данных Microsoft Access, расчеты производятся в Microsoft Excel, а отчет распечатывается в формате документа Microsoft Word. Кроме того, отчет надо представить в виде слайдов Microsoft Power на ежедневной планерке, а сводку о них опубликовать на Web-сайте. Оказывается всю эту комплексную задачу можно автоматизировать, причем, благодаря интегрированию. Управление можно производить из одного приложения, или даже из одного исполняемого файла, созданного на VB или VB .NET. В последнем случае пользователь даже не будет знать, что весь этот большой объем работы выполняется не данным приложением, а программными продуктами Microsoft Office.

Данная книга как раз и посвящена изучению этого замечательного языка программирования — VBA. Она является переработанным вторым изданием книги "Самоучитель VBA" издательства BHV-Петербург. С ее помощью вы получите базовые сведения о VBA, научитесь работать с макросами, освоите технологию объектно-ориентированного программирования, научитесь конструировать пользовательский интерфейс и формы, создавать защиту от дурака, работать с файлами, использовать и конструировать Web-службы. В Excel вы сумеете автоматизировать работу с рабочим листом, диаграммами и обработку данных. В Word вы научитесь создавать макросы, пользовательские стили и шаблоны. В Access узнаете, как создаются базы данных и производится работа с ними, как конструируется пользовательский интерфейс для доступа и обработки данных. В PowerPoint научитесь автоматизировать процесс создания презентаций. Узнаете, как в офисных проектах можно обрабатывать XML документы. Узнаете, как, используя Microsoft Visual Studio Tools для Microsoft Office System, в .NET приложения можно интегрировать офисные приложения. Научитесь создавать скрипты для Windows, а также применять в офисных приложениях COM компоненты и Win32 API (например, узнаете, как создать прозрачную форму и форму произвольного формата). VBA позволяет не только автоматизировать приложения, но и программировать саму среду разработки кода. Об этом также пойдет разговор в данной книге. Она научит вас программно экспортировать и удалять модули, импортировать в них текстовые файлы, как удалять, так и создавать в модулях код, создавать формы. Будет приведен пример программы, которая выполняется только один раз.

Книга насыщена большим числом примеров (около 350), которые выполняют двоякую функцию: во-первых, это готовый код, который вы сможете использовать в своих приложениях, а во-вторых, изучив эти примеры, вы с легкостью сможете самостоятельно создавать любые офисные приложения.

Для кого предназначена эта книга

Книга рассчитана на широкий круг читателей: от начинающего пользователя, который научится самостоятельно создавать офисные приложения, до эксперта, который найдет в ней обширные справочные сведения по технологии разработки подобных приложений. Благодаря тщательной подборке обширной коллекции примеров, она может также быть полезна как студентам, изучающим офисное программирование, так и преподавателям, ведущим занятия по офисному программированию.

Об авторе

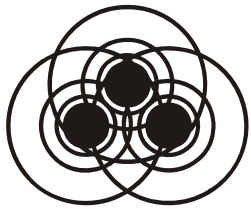
Андрей Юрьевич Гарнаев, доктор физико-математических наук, профессор Санкт-Петербургского государственного университета.

Область интересов: программное обеспечение, математическая кибернетика. Является автором десяти книг по программированию и программным средствам, двух — по теории игр, а также книг по офисному программированию: "Excel 2002: разработка приложений", "Excel, VBA и Internet в экономике и финансах", "Visual Basic .NET: разработка приложений", вышедших в издательстве BHV-Санкт-Петербург. Более подробную информацию об авторе можно найти на сайте **http://www.apmath.spbu.ru/~kmms/garnaev/default.htm**.

Данная книга написана на основе многолетнего опыта разработки офисных приложений и курса лекций, читаемого в Санкт-Петербургском государственном университете, а также с учетом многочисленных комментариев и пожеланий читателей, которым автор искренне признателен.

Благодарю вас за выбор этой книги. Надеюсь, ее чтение для вас будет не только полезным, но и увлекательным.

Глава 1



Введение

В данной главе пойдет разговор о том, что такое VBA, а также будет продемонстрировано, как при помощи встроенного средства макрорекордер можно автоматизировать и визуализировать процесс создания кода. Компьютеру достаточно показать, что вы хотите получить в конечном счете, а макрорекордер сам переведет эти действия в программный код. В главе также объясняется, как макросы можно назначать различным объектам и элементам управления, тем самым создавая индивидуальный интерфейс вашего приложения, описывается, где хранятся макросы и как сделать, чтобы они были достижимы любой рабочей книге.

Что такое VBA

Существует целый ряд систем программирования, позволяющих в той или иной степени реализовать концепцию объектно-ориентированного подхода при разработке программных средств. К ним относятся C++, Java, Visual Basic. В отличие от VB, VBA не является языком объектно-ориентированного программирования в строгом смысле этого слова, но в нем широко используются элементы объектно-ориентированного подхода и связанные с ним понятия.

Язык программирования Visual Basic for Application вначале стал применяться как средство, которое позволило Excel, а затем и другим приложениям Microsoft Office программно управлять их собственной средой. Оно взаимодействовало с другими приложениями, используя механизм автоматизации OLE.

Язык VBA — это подмножество VB, которое включает почти все его средства создания приложений, структуры данных и управляющие структуры, возможность создания пользовательских типов данных.

Visual Basic for Application, как и Visual Basic, является языком визуального и событийно управляемого программирования. Он реализует следующие возможности: создание нестандартного окна диалога в виде формы с базовым набором элементов управления, создание формы на рабочем листе, написание

процедур, обрабатывающих события, которые возникают при тех или иных действиях системы и конечного пользователя.

Язык VBA не обладает всеми возможностями VB, но позволяет работать с огромным набором объектов — по существу в нем определены все объекты приложений MS Office.

Отметим одну, может быть главную особенность программирования на VBA: создание проекта на каком-либо языке программирования осуществляется в соответствующей среде, где язык — главное средство, а создание кода (последовательности операторов) — главная цель действия программиста. А при работе на VBA целью является создание документа в широком смысле (документ Word, рабочая книга Excel, презентация, база данных Access). Проект (программа) на VBA — результат побочной деятельности по созданию документа. Более того, проект на VBA нельзя создать независимого от какого-либо документа, даже если никакие свойства этого документа не используются.

Приступая к очередному сеансу работы, программист открывает одно из приложений MS Office, и в этот момент в языке VBA автоматически становится доступным объект `Application`, определяющий это приложение, а также все встроенные в него объекты.

Итак, VBA отличается от VB и прочих языков программирования тем, что он предоставляет возможность непосредственной работы с объектами MS Office. Это позволяет эффективно его использовать для автоматизации деятельности, связанной с обработкой различных типов документов.

VBA позволяет существенно расширить вычислительные средства MS Office. С помощью VBA можно легко и быстро создавать различные приложения, даже не являясь программистом.

VBA имеет графическую инструментальную среду, позволяющую создавать экранные формы и управляющие элементы. С помощью VBA можно создавать собственные функции для Excel, вызываемые мастером функций, разрабатывать макросы, создавать собственные меню и многое другое. VBA позволяет с легкостью решать задачи, которые средствами Excel практически невозможно решить.

VBA реализует концепцию визуального программирования, управляемого событиями.

Макросы

Если вы осуществляете многократно повторяющиеся действия, то этот процесс можно автоматизировать при помощи макросов. *Макрос* (*macro*) — это последовательность команд, которые написаны на VBA и которые хранятся в стандартном модуле среды разработки VBA приложений. Когда возникает необходимость выполнить данную последовательность действий, пользова-

тель может запустить на выполнение соответствующий макрос. Макросы могут осуществлять широкий спектр задач от простых вычислений до создания пользовательского интерфейса вашего приложения. Их возможности ограничены только вашей фантазией и уровнем знания VBA. Нет необходимости писать макросы вручную. В MS Office имеется встроенное средство — *макрорекордер* (*macro recorder*), позволяющее преобразить все ваши действия в макросы. Их потом остается только немного отредактировать для придания большей функциональности и гибкости. Использование макрорекордера является замечательным подспорьем при изучении VBA.

Запись макроса

Прежде чем начать запись макроса, надо тщательно спланировать ваши действия. Давайте, для примера, запишем макрос, который создает новую рабочую книгу, состоящую из единственного рабочего листа **Отчет** с шаблоном отчетной таблицы, которую остается только заполнить (рис. 1.1).

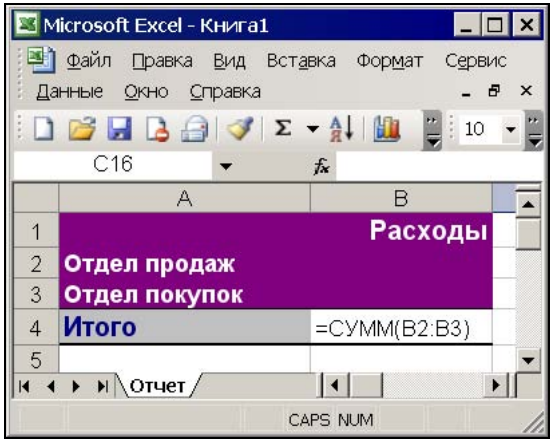


Рис. 1.1. Создаваемый шаблон таблицы

Для активизации макрорекордера выберите команду **Сервис | Макрос | Начать запись**. Появится диалоговое окно **Запись макроса** (рис. 1.2). Это диалоговое окно позволяет задать параметры макроса.

Поля **Имя макроса** и **Описание** используются для задания имени макроса и его описания. Описание важно для многократно используемых макросов. Наша память не долговечна, и, не имея подсказки в виде описания, через некоторое время будет трудно вспомнить, для чего тот или иной макрос создавался. По умолчанию макросам присваиваются имена `Макрос1`, `Макрос2` и т. д. С целью облегчения узнаваемости макроса лучше не использовать стандартное имя, а присвоить ему какое-нибудь уникальное имя, поясняющее,

для чего он используется. В нашем случае, например, присвоим макросу имя `СоздатьОтчет`. Поле **Сочетание клавиш** позволяет назначить макросу комбинацию клавиш, т. е. указать символ, который в комбинации с клавишей <Ctrl> позволить его выполнить. Назначать комбинацию клавиш макросу совсем не обязательно. Это стоит делать только для постоянно используемых макросов, для быстрого доступа к ним. Не используя комбинации клавиш, макрос можно всегда вызвать командой **Сервис | Макрос | Макросы**. Раскрывающийся список **Сохранить в** предназначен для выбора книги, в которой будет сохранен макрос. Если выбрать **Личная книга макросов**, то макрос будет сохранен в специальную скрытую книгу, в которой хранятся макросы. Эта книга всегда существует, хотя и скрыта, и записанные в ней макросы доступны для других рабочих книг. Команда **Окно | Отобразить** позволяет отобразить личную книгу макросов. Если в раскрывающемся списке выбрать **Эта книга** (т. е. выбор, который по умолчанию предлагает компьютер), то макрос сохранится на новом листе модуля в активной рабочей книге, а если выбрать **Новая книга**, то он сохранится в новой рабочей книге.

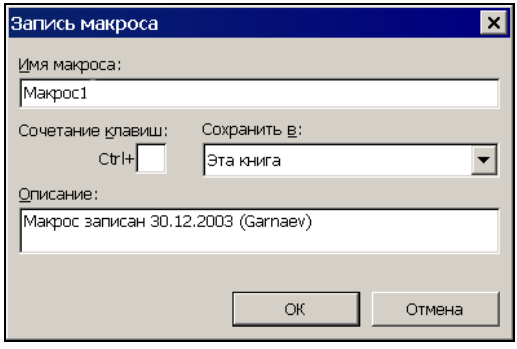


Рис. 1.2. Диалоговое окно **Запись макроса**

Итак, зададим параметры макроса в диалоговом окне **Запись макроса**:

- в поле **Имя макроса** введите `СоздатьОтчет`;
- в поле **Описание** введите `Создание рабочей книги с отчетной таблицей`;
- в поле **Сочетание клавиш** введите **а**. Нажмите кнопку **OK**.

Появится плавающая панель инструментов с кнопкой **Остановить запись**. Теперь все производимые действия будут записываться до тех пор, пока не будет нажата эта кнопка или не будет выбрана команда **Сервис | Макрос | Остановить запись**. Итак, запишем макрос:

- выберите команду **Сервис | Параметры**. На вкладке **Общие** окна **Параметры** в поле **Листов в новой книге** введите 1 и нажмите кнопку **OK**;
- выберите команду **Файл | Создать**. В окне **Создание книги** нажмите кнопку **Чистая книга**. В результате в проект будет добавлена новая книга;

- ❒ выберите ярлык листа **Лист1** этой книги и в контекстном меню выберите команду **Переименовать**. Измените имя листа на **Отчет**;
- ❒ выберите ячейку **A2** и введите в нее `Отдел продаж`;
- ❒ выберите ячейку **A3** и введите в нее `Отдел покупок`;
- ❒ выберите ячейку **A4** и введите в нее `Итого`;
- ❒ выберите столбец **A**, а затем команду **Формат | Столбец | Автоподбор ширины**;
- ❒ выберите ячейку **B1** и введите в нее `Расходы`;
- ❒ выберите ячейку **B4** и введите в нее при помощи кнопки **Автосумма** формулу

  ```
  =СУММ(B2:B3);
  ```

- ❒ для форматирования таблицы выделите диапазон **A1:B4**, а затем выберите команду **Формат | Автоформат**, и в окне **Автоформат** выберите **Классический 2**;
- ❒ Выберите команду **Сервис | Макрос | Остановить запись** для остановки записи.

Для просмотра только что записанной процедуры выберите команду **Сервис | Макрос | Макросы**, что приведет к отображению диалогового окна **Макрос** (рис. 1.3).

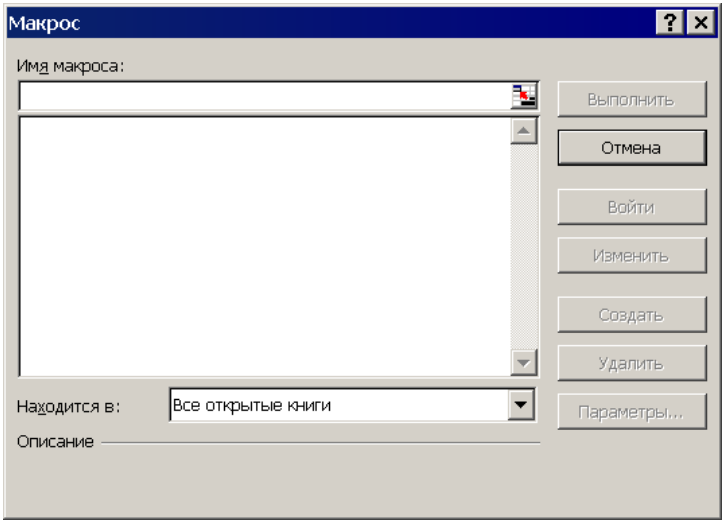


Рис. 1.3. Диалоговое окно **Макрос**

В диалоговом окне **Макрос** в списке выделите макрос и нажмите кнопку **Изменить**. На экране отобразится окно редактора VBA с активизированным стандартным модулем, в котором будет код (листинг 1.1) только что записанного макроса.

Листинг 1.1. Макрос, активизирующий рабочий лист Лист2. Стандартный модуль

```
Sub СоздатьОтчет()
'
' СоздатьОтчет Macro
' Создание рабочей книги с отчетной таблицей
'
    Application.SheetsInNewWorkbook = 1
    Workbooks.Add
    Sheets("Лист1").Select
    Sheets("Лист1").Name = "Отчет"
    Range("A2").Select
    ActiveCell.FormulaR1C1 = "Отдел продаж"
    Range("A3").Select
    ActiveCell.FormulaR1C1 = "Отдел покупок"
    Range("A4").Select
    ActiveCell.FormulaR1C1 = "Итого"
    Columns("A:A").Select
    Selection.Columns.AutoFit
    Range("B1").Select
    ActiveCell.FormulaR1C1 = "Расходы"
    Range("B4").Select
    ActiveCell.FormulaR1C1 = "=SUM(R[-2]C:R[-1]C)"
    Range("A1:B4").Select
    Selection.AutoFormat Format:=xlRangeAutoFormatClassic2, _
        Number:=True, Font:=True, Alignment:=True, _
        Border:=True, Pattern:=True, Width:=True
End Sub
```

Выполнение макроса

Для выполнения макроса:

- выберите команду **Сервис | Макрос | Макросы**;
- в окне **Макрос** выберите ссылку на макрос и нажмите кнопку **Выполнить**.

Макрос можно выполнить и при помощи горячих клавиш. В нашем случае такой комбинацией была <Ctrl>+<a>. Просто нажмите ее, и книга с шаблоном отчета будет создана.

Редактирование макроса

Созданный макрос можно редактировать, упрощая его код и придавая ему большую общность, и, если необходимо, интерактивность. В качестве примера давайте проанализируем созданный макрорекордером код и упростим его.

Первая инструкция говорит о том, что новая книга будет состоять из одного рабочего листа. Эту инструкцию оставим без изменения.

```
Application.SheetsInNewWorkbook = 1
```

Вторая инструкция создает новую рабочую книгу. Ее тоже оставим без изменения.

```
Workbooks.Add
```

Первая из следующих двух инструкция выбирает рабочий лист **Лист1**, а затем переименовывает его в **Отчет**.

```
Sheets("Лист1").Select
Sheets("Лист1").Name = "Отчет"
```

Эти две инструкции можно заменить одной. Во-первых, код будет короче, а во-вторых, он не будет зависеть от локальной версии MS Office, т. к. доступ к листу будет производиться не по его имени, а по номеру.

```
Sheets(1).Name = "Отчет"
```

Первая из следующих двух инструкций выбирает ячейку **A2**, а вторая вводит в выбранную ячейку значение `Отдел продаж`.

```
Range("A2").Select
ActiveCell.FormulaR1C1 = "Отдел продаж"
```

Эти две инструкции также лучше заменить одной, которая вводит значение непосредственно в ячейку **A2**.

```
Range("A2").Value = "Отдел продаж"
```

Следующие четыре инструкции вводят значения в ячейки **A3** и **A4**.

```
Range("A3").Select
ActiveCell.FormulaR1C1 = "Отдел покупок"
Range("A4").Select
ActiveCell.FormulaR1C1 = "Итого"
```

Их можно заменить двумя инструкциями:

```
Range("A3").Value = "Отдел покупок"
Range("A4").Value = "Итого"
```

Первая из следующих двух инструкций выбирает столбец **A**, а вторая автоматически подбирает у выбранного столбца ширину.

```
Columns("A:A").Select
Selection.Columns.AutoFit
```

Эти две инструкции можно заменить одной, которая непосредственно у столбца **A** подбирает ширину.

```
Columns("A:A").AutoFit
```

Следующие две инструкции вводят значение в ячейку **B1**.

```
Range("B1").Select
ActiveCell.FormulaR1C1 = "Расходы"
```

Их можно заменить одной инструкцией:

```
Range("B1").Value = "Расходы"
```

Первая из следующих двух инструкций выбирает ячейку **B4**, а вторая вводит в выбранную ячейку формулу `=СУММ(B2:B3)` в относительном R1C1 формате.

```
Range("B4").Select
ActiveCell.FormulaR1C1 = "=SUM(R[-2]C:R[-1]C)"
```

Их можно заменить одной инструкцией, вводящей непосредственно в ячейку **B4** данную формулу в A1 формате локальной версии.

```
Range("B4").FormulaLocal = "=СУММ(B2:B3)"
```

И последние две инструкции, реализующие форматирование таблицы

```
Range("A1:B4").Select
Selection.AutoFormat Format:=xlRangeAutoFormatClassic2, _
     Number:=True, Font:=True, Alignment:=True, _
     Border:=True, Pattern:=True, Width:=True
```

также можно заменить только одной

```
Range("A1:B4").AutoFormat Format:=xlRangeAutoFormatClassic2, _
               Number:=True, Font:=True, Alignment:=True, _
               Border:=True, Pattern:=True, Width:=True
```

В заключении соберем все сделанные изменения в листинге 1.2.

Листинг 1.2. Упрощенный код макроса

```
Sub СоздатьОтчет()
   Application.SheetsInNewWorkbook = 1
   Workbooks.Add
   Sheets(1).Name = "Отчет"
   Range("A2").Value = "Отдел продаж"
   Range("A3").Value = "Отдел покупок"
   Range("A4").Value = "Итого"
   Columns("A:A").AutoFit
   Range("B1").Value = "Расходы"
   Range("B4").FormulaLocal = "=СУММ(B2:B3)"
   Range("A1:B4").AutoFormat Format:=xlRangeAutoFormatClassic2, _
```

```
                    Number:=True, Font:=True, Alignment:=True, _
                    Border:=True, Pattern:=True, Width:=True
End Sub
```

Назначение макроса объекту

Назначить макрос можно любому объекту, расположенному на рабочем листе или в меню вашей книги. Например, назначим его кнопке:

- ❒ отобразите панель инструментов **Формы**;
- ❒ расположите элемент управления **Кнопка (Button)** на рабочем листе. Щелкните на нем правой клавишей и измените тест, отображаемый на его поверхности с `Кнопка1` на `Отчет`;
- ❒ из контекстного меню, ассоциированного с кнопкой, выберите команду **Назначить макрос**. В окне **Назначить макрос объекту** выберите ссылку на исходный макрос и нажмите кнопку **ОК** (рис. 1.4).

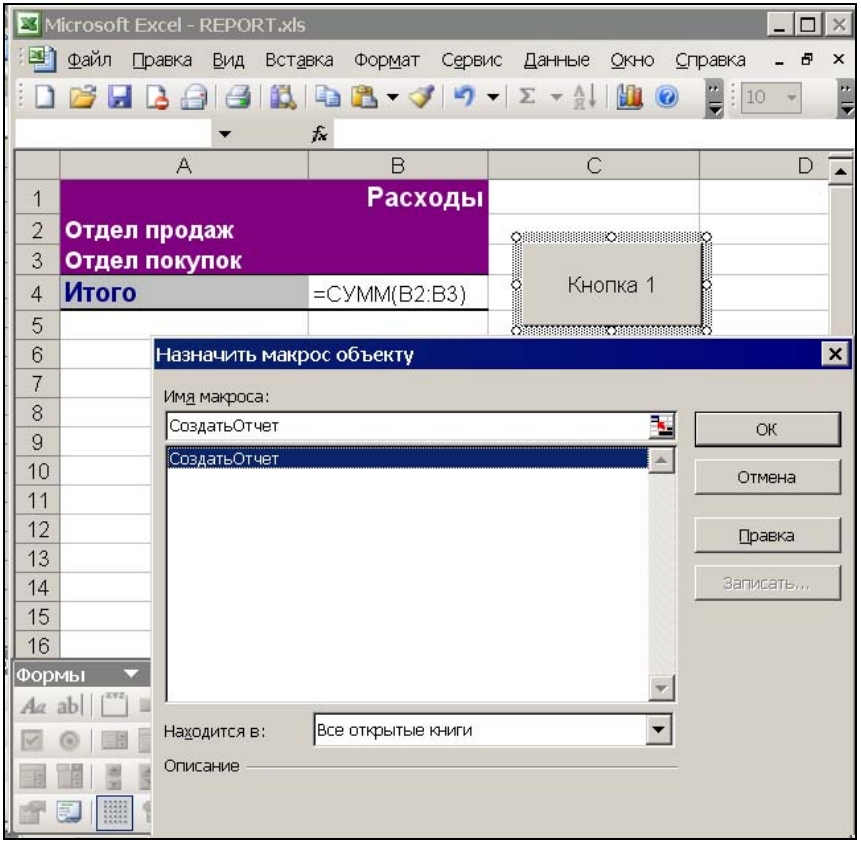


Рис. 1.4. Кнопка, панель инструментов **Формы** и окно **Назначить макрос объекту**

Аналогично макрос можно назначить любому графическому объекту, например автофигуре, рисунку или WordArt объекту.

Как записывать относительную ссылку на ячейки

По умолчанию все ссылки на диапазоны, записываемые макрорекордером, являются абсолютными. Это означает, что диапазон первоначально специфицируется, а затем над ним совершается действие, например, в следующем коде диапазон **A1** специфицируется методом `Select`, а затем его содержимое копируется в буфер обмена. Такой подход ограничен в силу того, что при повторном выполнении кода то же самое действие будет выполнено над тем же диапазоном.

```
Range("A1").Select
Selection.Copy
```

Для создания относительной ссылки перед началом записывания действий надо нажать кнопку **Относительная ссылка**. Например, запишем в относительных ссылках следующие действия:

- выберите ячейку **A1** и введите в нее `a`;
- выберите ячейку **A2** и введите в нее `b`;
- выберите ячейку **A3** и введите в нее `c`;
- выберите ячейку **B1** и введите в нее `3`;
- выберите ячейку **B2** и введите в нее `4`;
- выберите ячейку **B3** и введите в нее `=B1+B2`.

В результате будет записан следующий макрос:

```
Sub RelativeRef()
'
' RelativeRef Macro
' Macro recorded 25.11.2003 by Garnaev
'


'
    Range("A1").Select
    ActiveCell.Select
    ActiveCell.FormulaR1C1 = "a"
    ActiveCell.Offset(1, 0).Range("A1").Select
    ActiveCell.FormulaR1C1 = "b"
    ActiveCell.Offset(1, 0).Range("A1").Select
```

```
    ActiveCell.FormulaR1C1 = "c"
    ActiveCell.Offset(-2, 1).Range("A1").Select
    ActiveCell.FormulaR1C1 = "3"
    ActiveCell.Offset(1, 0).Range("A1").Select
    ActiveCell.FormulaR1C1 = "4"
    ActiveCell.Offset(1, 0).Range("A1").Select
    ActiveCell.FormulaR1C1 = "=R[-2]C+R[-1]C"
    ActiveCell.Offset(1, 0).Range("A1").Select
End Sub
```

Все ссылки в этом макросе, за исключением ссылки в первой инструкции, относительны по отношению к активной ячейке. Если эту инструкцию `Range("A1").Select` удалить, то в верхнем правом углу таблицы будет располагаться активная ячейка, а не ячейка **A1**.

Где хранятся макросы

Если вы хотите, чтобы ваш макрос был достижим в любой активной книге, то его надо хранить в файле Personal.xls. Этот файл создается по умолчанию и хранится в XLSTART каталоге. Для записи макроса в файл Personal.xls при инициализации макроса в окне **Макрос** в списке **Находится в** выберите **Все открытые книги**. Отметим, что по умолчанию файл Personal.xls является скрытым файлом.

Защита от макросов

MS Excel можно защитить от несанкционированных макросов. Для этого достаточно выбрать команду **Сервис | Макрос | Безопасность**. На экране отобразится окно **Безопасность.** На вкладке **Уровень безопасности** этого окна имеются четыре переключателя, которые управляют уровнем безопасности:

- переключатель **Очень высокая** — разрешается запуск только макросов, установленных в надежных расположениях. Все остальные подписанные и неподписанные макросы отключаются. Для полного отключения всех макросов можно задать уровень безопасности **Очень высокая** и отключить макросы, установленные в надежных расположениях. Чтобы отключить макросы, установленные в надежных расположениях, выберите в меню **Сервис** команды **Макрос и Безопасность**, а затем перейдите на вкладку **Доверенные источники** и снимите флажок **Доверять всем установленным надстройкам и шаблонам**. При этом также будут отключены все надстройки COM, DLL-файлы смарт-тегов и макросы;
- переключатель **Высокая** — допускаются только подписанные макросы из известных источников. MS Office XP использует MS Authenticode технологию

для создания цифровой подписи, которая идентифицирует автора макроса и его источник, а также подтверждает, что макрос не был изменен. Все остальные макросы автоматически отключаются, и загружается рабочая книга;

- переключатель **Средняя** — пользователю предоставляется выбор, отключать или нет макросы;
- переключатель **Низкая** — рабочая книга загружается без проверки макросов.

Элемент управления *Кнопка* и создание процедуры обработки события, при которой выполняется макрос

В VBA принят объектно-ориентированный подход разработки приложений. В первых версиях MS Office этот подход не был четко сформулирован, но в последующем, с появлением панели инструментов **Элементы управления**, этот подход стал основным. Сразу же стоить отметить, что в Excel существуют две похожие панели инструментов **Элементы управления** и **Формы**. Панель инструментов **Формы** унаследована от версии Excel 5.0 и в основном используется для совместимости приложений, созданных в старых версиях.

Итак, нашей целью является создание приложения, в котором имеется кнопка **Отчет**, нажатие на которую приводит к созданию новой книги с шаблоном отчетной таблицы.

- Прежде всего, переведите Excel в режим конструктора элементов управления. Для этого нажмите кнопку **Режим конструктора**  панели инструментов **Элементы управления**.
- Теперь можно нажать кнопку  и нарисовать элемент управления **Кнопка**, точно так же, как и при помощи любого графического редактора, в том месте рабочего листа, где вы задумали расположить кнопку. Как и любой графический объект, кнопку можно рисовать при нажатой клавише <Shift>, что обеспечит ей квадратную форму, либо при нажатой клавише <Alt> — для привязки кнопки к сетке рабочего листа. У созданной кнопки при помощи маркеров изменения размеров можно изменять размеры, а при помощи маркера перемещения — изменять ее местоположение.
- На поверхности первого созданного элемента управления **Кнопка** автоматически будет отображена надпись **CommandButton1**. Если вы создадите второй элемент управления **Кнопка**, то на его поверхности отобразится надпись **CommandButton2** и т. д. Элемент управления **Кнопка** является объектом, т. е. он, как и любой объект, обладает свойствами, методами и событиями. Надпись, отображаемая на поверхности элемента управления

Кнопка, задается значением свойства `Caption`. Кроме того, элемент управления **Кнопка** имеет свойство `Name`, которое в коде идентифицирует его как объект. В данном случае, оно тоже равно `CommandButton1`. Изменим значение свойства `Caption`, т. е. надпись, отображаемую на поверхности кнопки, с **CommandButton1** на **Отчет** (с тем, чтобы подчеркнуть, что эта кнопка будет создавать отчет). Для этого нажмите кнопку **Свойства**  панели инструментов **Элементы управления**. На экране отобразится окно **Properties** (рис. 1.5). Введите в поле `Caption` этого окна значение `Отчет`. Кроме того, измените значение свойства `Name` кнопки с неинформативного `CommandButton1` на `cmdReport`. Закройте окно **Properties**.

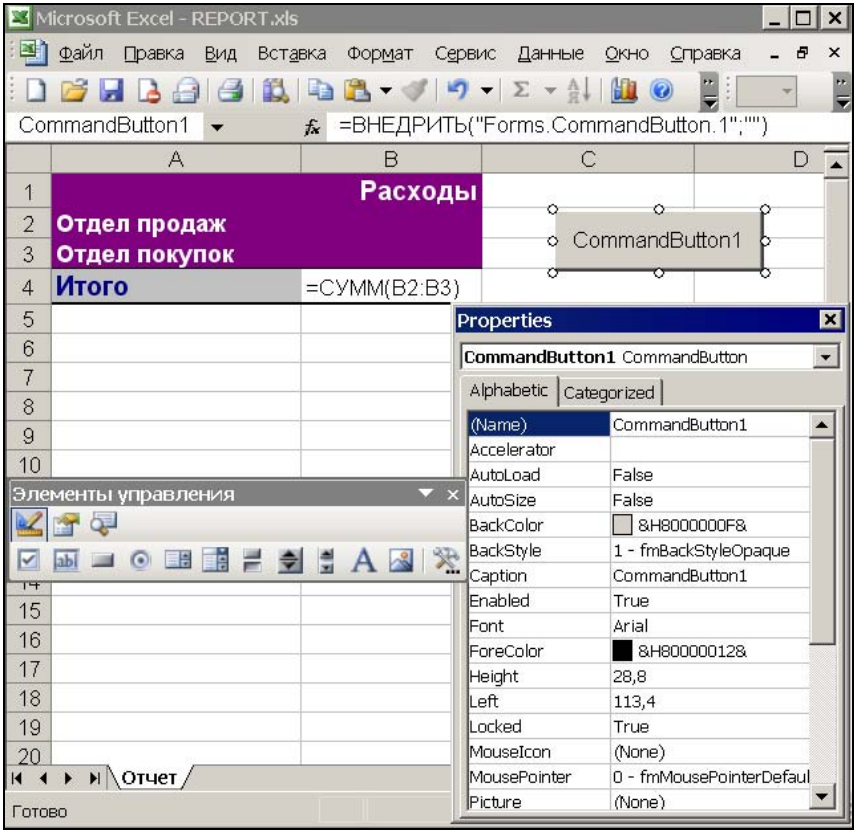


Рис. 1.5. Кнопка, панель инструментов **Элементы управления** и окно **Properties**

- Теперь можно перейти к созданию кода процедуры, обрабатывающей событие "нажатие на кнопку". Для этого дважды щелкните на созданной кнопке (напоминаем, что все это еще происходит в режиме конструктора). В результате откроется редактор VBA с активизированным модулем

рабочего листа (в данном случае, **Отчет**). Кроме того, двойной щелчок на кнопке создаст в модуле первую и последнюю инструкции процедуры обработки события `Click` — "нажатие на кнопку" (листинг 1.3, *а*).

Листинг 1.3, *а*. Первая и последняя инструкции процедуры обработки события "нажатие на кнопку". Модуль рабочего листа Лист1

```
Private Sub cmdReport_Click()


End Sub
```

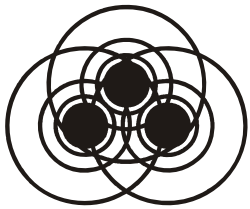
- Вот теперь, нам пригодится ранее созданный макрос (см. листинг 1.2). Через буфер обмена скопируйте имя макроса в процедуру обработки события "нажатие на кнопку" (листинг 1.3, *б*). Конечно, ее можно было бы набрать и вручную. Но это довольно медленно и чревато появлением случайных опечаток, неизбежно время от времени тут и там появляющихся при наборе кода.

Листинг 1.3, *б*. Процедура обработки события "нажатие на кнопку", при котором будет выполнен макрос

```
Private Sub cmdReport_Click()
   СоздатьОтчет
End Sub
```

Итак, процедура создана. Осталось только вернуться на рабочий лист **Отчет**. Созданная кнопка будет обрабатывать событие (нажатие на нее) только после выхода из режима конструктора. Поэтому отключите режим конструктора нажатием кнопки **Режим конструктора**  панели инструментов **Элементы управления**. Теперь можно и протестировать созданную кнопку. Нажмите на нее, и, если вы все правильно сделали, то это приведет к созданию новой рабочей книги с листом **Отчет**, содержащей шаблон отчетной таблицы.

Глава 2



Основы программирования на VBA

В данной главе кратко пойдет разговор об основах программирования на VBA, о синтаксисе этого языка, о типах данных, переменных, массивах, перечисляемых типах и типах данных, определенных пользователем, преобразовании типов, форматировании данных, встроенных диалоговых окнах, операторах и процедурах языка, процедурах с необязательными параметрами, использующих неопределенное количество параметров, рекурсивных процедурах.

Типы данных

Типы данных относятся к фундаментальным понятиям любого языка. Тип данных определяет множество допустимых значений, которое может принимать указанная переменная. В VBA имеются следующие типы данных: `Byte` (байт), `Boolean` (логический), `Integer` (целое), `Long` (длинное целое), `Single` (с плавающей точкой обычной точности), `Double` (с плавающей точкой двойной точности), `Currency` (денежный), `Decimal` (масштабируемое целое), `Date` (даты и время), `Object` (объект), `String` (строка), `Variant` (тип используется по умолчанию), тип данных, определяемый пользователем, а также специфические типы объектов.

Переменные

В VBA *переменная* (*variable*) используется для временного хранения данных в оперативной памяти, т. е. она идентифицируют область памяти, где хранится некоторая информация. После объявления переменная указывает на одну и ту же область памяти до тех пор, пока не будет уничтожена. Программист не должен указывать явное место хранение данных, для ссылки на область памяти ему достаточно указать лишь имя. Переменная должна быть объявлена прежде, чем ее можно использовать. Объявление производится при помощи операторов `Dim`, `Private`, `Static` или `Public`, которые также определяют

и область видимости переменной. Например, следующая инструкция объявляет целую переменную (тип `Integer`).

```
Dim N As Integer
```

Если тип данных при объявлении переменной опущен, то по умолчанию переменная получает тип `Variant`. Например, следующая инструкция объявляет переменную `x` типа `Variant`.

```
Dim x
```

Определение времени жизни и области видимости переменных

Термины *время жизни и область видимости переменной* означают место использования переменной в приложении, а также время существования переменной после ее создания.

Область видимости переменной определяет часть кода, которая "знает" о существовании данной переменной. При определении переменной в процедуре получить или изменить ее значение можно только из кода этой процедуры. Иногда, однако, необходимо использовать переменную с более обширной областью видимости. Например, переменную, значения которой доступны для всех процедур модуля или проекта. Существуют три типа области видимости переменной:

- переменные уровня процедуры распознаются только в процедуре, в которой они описаны. Они описываются при помощи инструкций `Dim` или `Static`. Такие переменные называются *локальными*;
- переменные уровня модуля используются только в модуле, в котором они описаны, но не в других модулях данного проекта. Описываются при помощи оператора `Dim` или `Private` в области описания модуля, т. е. перед описанием процедур;
- переменные уровня модуля, описанные при помощи инструкции `Public`, являются доступными для всех процедур проекта. Такие переменные называются *открытыми*.

Закрытая (*`private`*) переменная сохраняет свое значение, только пока выполняется процедура, в которой эта переменная описана. При завершении процедуры значение переменной теряется, и при повторном запуске процедуры его надо заново инициализировать. Переменные, описанные оператором `Static`, сохраняют свое значение по выходу из процедуры, пока работает программа.

Установка необходимости определения переменных с помощью директивы *Option Explicit*

Для обязательного объявления всех переменных в начале модуля, в так называемой области модуля **General Declarations**, надо поместить директиву `Option Explicit`. Использование этой директивы не допускает возможности неправильного ввода имени переменной, которая применяется в одной или нескольких процедурах модуля. Например, если переменная была объявлена как `Ставка`, а в коде при наборе вместо русской буквы `С` была использована латинская буква `C`, то это приведет к ошибке. В отсутствие директивы `Option Explicit` подобную ошибку было бы трудно отследить.

Ключевое слово *Null*

Данные типа `Variant` могут иметь особое значение `Null`, которое означает, что данные отсутствуют, неизвестны или неприменимы. Например, по умолчанию данные в полях таблицы базы данных имеют тип `Variant`. Поэтому, если оставить поле пустым, ему будет присвоено значение `Null`. Функция `IsNull` проверяет, является ли указанное значение `Null`.

Комментарии

Текст, следующий в программе за символом (') вплоть до конца строки, игнорируется компилятором и представляет собой *комментарий*. Комментарии позволяют добавлять описания и пояснения для тех программистов, которые в будущем будут разбираться в вашей программе. Комментарии также полезны при отладке программ. Они позволяют временно отключать строки кода программы на этапе отладки. Приведем возможные варианты использования комментариев в коде программы (листинг 2.1).

Листинг 2.1. Комментарии

```
Dim a As Integer
' *************************
' *  a — целая переменная  *
' *************************
Dim b As String ' b — строковая переменная
' b = sin(2) — этот оператор отключен
```

Перенос строки кода

Расположение символов <Пробел>+<Знак подчеркивания> в конце строки обеспечивает то, что последующая строка является продолжением предыдущей. В следующем примере первая из конструкций является разбиением второй на две строки:

```
ActiveCell.Offset(rowoffset:=0, _
                  columnoffset:=1).Value = Сумма
```

и

```
ActiveCell.Offset(rowoffset:=0, columnoffset:=1).Value = Сумма
```

Строки

Строка (`string`) представляет собой последовательность символов, каждый из которых имеет тип `Char`. Последовательность символов, присваиваемая строковой переменной, должна быть окружена кавычками.

```
Dim str As String
str = "Это строка"
```

В VBA имеется единственная строковая операция — *конкатенация*. Эта операция применяется для объединения нескольких строк в одну. Операция конкатенация обозначается символом амперсанта "&" или сложения "+". В следующем примере переменной `s` присваивается значение "`Visual Basic for Applications`"

```
Dim s As String
s = "Visual Basic " & "for Applications"
```

Ключевое слово `Empty` возвращает ссылку на пустую строку. Того же эффекта можно добиться, поставив рядом двое кавычек (""). Например, следующий оператор отобразит окно с сообщением `Равносильны`.

```
If Empty = "" Then MsgBox "Равносильны"
```

Даты

Тип `Date` подразумевает как время, так и дату. Даты хранятся в виде чисел с плавающей десятичной точкой и позволяют отобразить даты из интервала от 1 января 100 года до 31 декабря 9999 года, а время из интервала от 0:00:00 до 23:59:59. Значения дат могут быть представлены в любом распознаваемом формате и должны быть окаймлены знаками "#". Например, следующие две инструкции присваивают переменным `d1` и `d2` одно и тоже значение — 31 января 2003 г.

```
Dim d1 As Date, d2 As Date
```

```
d1 = #1/31/2003#
d2 = #31 Jan 2003#
```

Когда числовое значение преобразуется в тип `Date`, целая часть числа инкапсулирует информацию о дате, а именно, количество дней, прошедших с 30 декабря 1899 года. Поэтому даты могут реализовываться как положительными, так и отрицательными числами, причем отрицательные значения указывают на более ранние даты. Дробная часть представляет собой время, как часть дня, например, 0.5 — это полдень.

Массивы

Как и в других языках программирования, в VBA вы можете использовать *массивы*. Массив, как и любую переменную, надо объявлять, используя операторы `Dim`, `Static`, `Private` и `Public`, которые также задают область видимости переменной. В массиве допускается описание до 60 размерностей. При определении размерности надо указывать верхнюю, а также нижнюю границу. Если нижний индекс не задан явно, нижняя граница массива определяется директивой `Option Base`. Если отсутствует директива `Option Base`, нижняя граница массива равняется нулю. Например, в следующем операторе объявляется одномерный массив (вектор) из 12 целых чисел, причем по умолчанию первый элемент массива — `A(0)`, а последний — `A(11)`. В этом случае говорят, что 0 — базовый индекс.

```
Dim A(11) As Integer
```

Данный же оператор объявляет двухмерный массив 3×3 (матрицу), состоящий из действительных чисел.

```
Dim B(2, 2) As Single
```

Можно изменить базовый индекс, написав в области объявлений модуля директиву `Option Base 1`. После этого индексы массивов `A` и `B` будут начинаться с единицы. Например, в следующем операторе объявляется вектор, состоящий из 11 элементов.

```
Option Base 1
Dim A(11) As Integer
```

Другим способом изменения базового индекса является использование ключевого слова `To` при объявлении массива.

```
Dim B(1 To 3, 1 To 3) As Single
Dim A(1 To 12) As Integer
```

Инициализацию элементов массива можно производить по-разному:

- последовательностью операторов:

```
Dim B(1, 1) As Single
B(0, 0) = 2 : B(0, 1) = 4
B(1, 0) = 1 : B(1, 1) = 6
```

- оператором цикла:

```
Dim M(1 To 9, 1 To 9) As Integer
Dim i As Integer
Dim j As Integer
For i = 1 To 9
   For j = 1 To 9
      M(i, j) = i * j
   Next
Next
```

Удобным способом определения одномерных массивов является функция `Array`, преобразующая список элементов, разделенных запятыми, в вектор из этих значений и присваивающая их переменной типа `Variant`. Допустима инициализация, как одномерного массива, так и многомерного, за счет применения вложенных конструкций с функциями `Array`.

- Инициализация одномерного массива:

```
Dim num As Variant
Dim s As Double
num = Array(10, 20)
s = num(0) + num(1)
MsgBox s
```

- Инициализация многомерного массива:

```
Dim CityCountry As Variant
CityCountry = Array(Array("Санкт-Петербург", "Россия"), _
                    Array("Кейптаун", "ЮАР"))
MsgBox CityCountry(0)(0)
' Отобразится Санкт-Петербург
MsgBox CityCountry(0)(1)
' Отобразится Россия
```

Динамические массивы

Иногда в процессе выполнения программы требуется изменять размер массива. В этом случае первоначально массив объявляют как динамический. Для этого при объявлении массива не надо указывать размерность, например:

```
Dim R() As Single
```

Затем в программе следует вычислить необходимый размер массива в некоторой переменной, например *n*, и изменить размер динамического массива с помощью оператора `ReDim`. В следующем примере сначала объявляется динамический массив, а затем устанавливаются границы его индекса.

```
Dim R() As Double
ReDim R(1 To 10)
```

Допустимо повторное использование инструкции `ReDim` для изменения числа элементов и размерностей массива. В следующем примере создается массив с результатами бросания монеты (листинг 2.2). Монета бросается до тех пор, пока три раза не выпадет орел. Размерность динамического массива после каждого броска корректируется с сохранением в нем ранее записанных результатов бросания монеты за счет использования ключевого слова `Preserve`.

Листинг 2.2. Изменение размерности массива

```
Dim Attempt()
Dim i As Integer
Dim score As Integer
Dim coin As Integer
i = 0
score = 0
Do
   i = i + 1
   coin = Int(2 * Rnd())
   ' 0 - решка
   ' 1 - орел
   If coin = 1 Then score = score + 1
   ReDim Preserve Attempt(i)
   Attempt(i) = coin
Loop Until score = 3
```

Определение границ индексов массива

Функции `LBound`, `UBound` возвращают минимальное и максимальное допустимые значения указанного индекса массива. Например, в следующем коде отобразится 100 и 5.

```
Dim A(1 To 100, 0 To 5)
MsgBox UBound(A, 1) & vbCr & UBound(A, 2)
```

Следующие инструкции позволяют перебирать элементы массива без указания его размерности (листинг 2.3).

Листинг 2.3. Перебор элементов массива

```
Dim d As Variant, i As Integer
d = Array("Пн", "Вт", "Ср", "Чт", "Пт")
For i = LBound(d) To UBound(d)
   MsgBox d(i)
Next
```

Константы

Константы в отличие от переменных не могут изменять свои значения. Использование констант делает программы легко читаемыми и проще исправляемыми, т. к. отпадает необходимость многократно заменять значения по тексту программы, достаточно лишь ввести новое значение при объявлении константы. Константы объявляются при помощи ключевого слова `Const`, перед которым может стоять модификатор доступа `Public` или `Private`. Например, следующие инструкции объявляют числовую и строковую константы:

```
Private Const Rate As Single = 0.2
Public Const FirmName = ""
```

Перечисляемый тип

Перечисляемый тип предоставляет удобный способ работы с константами и позволяет ассоциировать их значения с именами. Этот тип определяется при помощи ключевого слова `Enum`, перед которым может идти модификатор доступа `Public` или `Private`. В следующем примере задается перечисляемый тип для идентификации стороны монеты (листинг 2.4).

Листинг 2.4. Перечисляемый тип

```
Enum Coin
   Head = 1
   Tail = -1
End Enum

Sub Attemp()
   Dim r As Integer
   Randomize
   r = 2 * Int(2 * Rnd()) - 1
   Select Case r
   Case Head
      MsgBox "Орел"
   Case Tail
      MsgBox "Решка"
   End Select
End Sub
```

Если перечисляемым константам (в данном случае `Head` и `Tail`) явно не заданы значения, то по умолчанию они полагаются равными 0, 1, .

Тип данных, определенный пользователем

Наряду с массивами, представляющими нумерованный набор элементов одного типа, существует еще один способ создания структурного типа — определенный пользователем тип или, в привычной терминологии для программистов — *запись*. Запись — это совокупность нескольких элементов, каждый из которых может иметь свой тип. Элемент записи называется полем. Запись является частным случаем класса, в котором не определены свойства и методы. Данный тип определяется при помощи ключевого слова `Type`, перед которым может идти модификатор доступа `Public` или `Private`. Инструкция `Type` используется только на уровне модуля. Появлению в модуле класса инструкции `Type` должно предшествовать ключевое слово `Private`. В следующем примере (листинг 2.5) инструкция `Type` используется для определения типа данных `MyType`, инкапсулирующего в себе информацию о персонажах сказок. У этой записи имеются три поля. Первая имеет тип `Integer` и содержит идентификационный номер героя, вторая имеет тип `String`, и специфицирует его имя. Третья имеет тип `Variant` и может содержать любой тип данных, и даже динамический массив.

Листинг 2.5. Тип, определенный пользователем

```
Private Type MyType
   ID As Integer
   Name As String
   Info As Variant
End Type

Sub TestMyType()
   Dim a(1) As MyType
   Dim txt() As String
   ReDim txt(1)
   txt(0) = "Smith"
   txt(1) = "From Wonderland"
   a(0).ID = 2
   a(0).Name = "Alice"
   a(0).Info = txt

   ReDim txt(2)
   txt(0) = "Pooh"
   txt(1) = "Bear"
   txt(2) = "C. Robin's friend"
```

```
    a(1).ID = 1
    a(1).Name = "Winnie"
    a(1).Info = txt

    Debug.Print "   a0   "
    Debug.Print a(0).ID & " " & a(0).Name _
          & " " & a(0).Info(0) & " " & a(1).Info(1)
    Debug.Print "   a1   "
    Debug.Print a(1).ID & " " & a(1).Name _
          & " " & a(1).Info(0) & " " & a(1).Info(1) & " " & a(1).Info(2)
End Sub
```

Математические операции

В VBA поддерживается стандартный набор математических операций от сложения до возведения в степень. Следующий код демонстрирует их применение.

```
Dim x As Double, y As Double, z As Double
x = 1 : y = 3
z = x + y ' Сложение
z = x - y ' Вычитание
z = x * y ' Умножение
z = x / y ' Деление
z = x \ y ' Целочисленное деление
z = x Mod y ' Остаток от деления по модулю
z = x ^ y ' Возведение в степень
```

Операции отношения

В VBA используется стандартный набор операций отношения. Следующий код демонстрирует их применение.

```
Dim x As Double, y As Double, b As Boolean
x = 1 : y = 3
b = (x < y) ' Меньше
b = (x <= y) ' Меньше или равно
b = (x > y) ' Больше
b = (x >= y) ' Больше или равно
b = (x <> y) ' Не равно
b = (x = y) ' Равно
```

Логические операции

Перечислим в табл. 2.1 логические операции, используемые в VBA.

Таблица 2.1. *Логические операции*

Операция	Описание
exp1 And *exp2*	Логическое умножение
exp1 Or *exp2*	Логическое сложение
exp1 Xor *exp2*	Исключающее Or, т. е. возвращает True тогда и только тогда, когда только один операнд возвращает True
exp1 Not *exp2*	Логическое отрицание
exp1 Imp *exp2*	Логическая импликация (в настоящее время почти не используется)
Exp1 Equ *exp2*	Логическая эквивалентность (в настоящее время почти не используется)

Директива *Option Compare*

Действие операции сравнения Like зависит от директивы Option Compare, которая располагается в области описания модуля. По умолчанию для каждого модуля считается установленной инструкция Option Compare Binary, при которой различаются строчные и прописные буквы, т. е. следующий оператор вернет False:

```
Debug.Print "AA" Like "aa"
```

Если же в области описания модуля указана инструкция Option Compare Text, то при сравнении строчные и прописные буквы не различаются, и тот же самый оператор на этот раз вернет True.

В следующем примере (листинг 2.6) в поле ввода диалогового окна пользователь должен ввести свое имя латинскими буквами. При нажатии кнопки **OK** программа анализирует информацию и отображает сообщение. А именно:

- если пользователь забыл ввести имя, то его об этом информируют;
- если во введенном имени присутствуют знаки, отличные от букв латинского алфавита, его об этом информируют;
- если имя состоит только из букв латинского алфавита, то с ним здороваются.

Листинг 2.6. Операция сравнения `Like`

```
Option Compare Text

Sub DemoCompare()
   Dim name As String, lng As Integer, i As Integer
   name = InputBox("Введите имя")
   lng = Len(Trim(name))
   If lng = 0 Then
      MsgBox "Забыли ввести имя"
      Exit Sub
   Else
      For i = 1 To lng
         If Not Mid(name, i, 1) Like "[A-Z]" Then
            MsgBox "Имя состоит только" & vbCr _
                   & "из букв латинского алфавита"
            Exit Sub
         End If
      Next i
   End If
   MsgBox " Привет, " & name
End Sub
```

Математические функции

В VBA имеется большой список математических функций от нахождения косинуса числа до генерации случайного числа, позволяющих произвести любые вычисления. В табл. 2.2 перечислены математические функции VBA.

Таблица 2.2. *Математические функции VBA*

Функция	Описание
`Abs`	Модуль (абсолютная величина) числа
`Atn`	Арктангенс
`Cos`	Косинус
`Exp`	Экспонента, т. е. результат возведения основания натурального логарифма в указанную степень
`Log`	Натуральный логарифм

Таблица 2.2 *(окончание)*

Функция	Описание
Rnd	Возвращает очередное число из последовательности псевдослучайных чисел из интервала [0, 1]. Rnd(*number*) Если значение параметра *number* меньше нуля, то Rnd возвращает каждый раз одно и то же число. Если значение параметра *number* больше нуля или этот параметр опущен, то Rnd возвращает следующее случайное число в последовательности. Если значение параметра *number* равняется нулю, то Rnd возвращает случайное число, возвращенное при предыдущем вызове этой функции
Sgn	Знак числа
Sin	Синус
Sqr	Квадратный корень из числа
Tan	Тангенс
Fix, Int	Обе функции Int и Fix отбрасывают дробную часть числа и возвращают целое значение. Различие между функциями Int и Fix состоит в том, что для отрицательного значения параметра функция Int возвращает ближайшее отрицательное целое число, меньшее либо равное указанному, а Fix — ближайшее отрицательное целое число, большее либо равное указанному

Создание последовательности случайных чисел

Функция Rnd генерирует очередное число из последовательности псевдослучайных чисел. У таких последовательностей, если совпадают первые члены, то и совпадают все последующие. Перед вызовом функции Rnd следует применять оператор Randomize, который определяет первый член этой последовательности. Если этот оператор используется без параметра, то первый член последовательности привязан к текущему системному времени, что делает число, возвращаемое функцией Rnd, "более" случайным. Например, в листинге 2.7 генерируются три случайных числа из интервала от 0 до 1.

Листинг 2.7. Последовательность случайных чисел

```
Dim i As Integer
Randomize
For i = 1 To 3
   Debug.Print Rnd()
Next
```

Как найти значение числа π

В VBA нет функции, возвращающей число π. Поэтому для нахождения числа π применяется функция `Atn`, как показано в следующем примере:

```
Dim Pi As Double
Pi = 4 * Atn(1)
```

Конечно, можно задать π и явным образом, указав достаточное число значащих цифр, но этот подход менее элегантен, чем с помощью функции.

Функции проверки типов

Функция проверки типа проверяет, является ли переменная выражением специфицированного типа. В табл. 2.3 приведены используемые в VBA функции проверки типов.

Таблица 2.3. *Функции проверки типов*

Функция	Проверяемый тип
`IsArray`	Является ли значение переменной массивом
`IsDate`	Является ли значение переменной датой
`IsEmpty`	Была ли переменная явно объявлена
`IsError`	Является ли код ошибки значением переменной
`IsNull`	Является ли `Null` значением переменной
`IsNumeric`	Является ли число значением переменной
`IsObject`	Является ли объект значением переменной

Преобразование строки в число

Функция `Val` возвращает подходящее число, содержащееся в строке, указанной в качестве значения ее параметра. В качестве допустимого десятичного разделителя функция `Val` воспринимает только точку. При наличии другого десятичного разделителя (например, запятой) для преобразования чисел в строки следует использовать функцию `CStr`, описанную далее. Код из листинга 2.8 демонстрирует результаты преобразования строки в число.

Листинг 2.8. Примеры преобразования строки в число

```
Dim s As String
s = "23432"
Debug.Print Val(s) ' Отобразится 23432
```

```
s = "123-45-45"
Debug.Print Val(s) ' Отобразится 123
s = #12/15/1999#
Debug.Print Val(s) ' Отобразится 15.12
s = "198,005"
Debug.Print Val(s) ' Отобразится 198
s = "198.005"
Debug.Print Val(s) ' Отобразится 198.005
```

Преобразование числа в строку

Функция `CStr` возвращает значение типа `Variant` (`String`), являющееся строковым представлением числа. Функция `CStr` преобразует значение в строку, т. е. в значение типа `String`. Например, код из листинга 2.9 проверяет, является ли 0 одной из цифр сгенерированного целого случайного числа из интервала от 1 до 1000. Для этого число преобразуется в строку, а затем с помощью функции `InStr` проверяется, является ли указанная строка подстрокой данной.

Листинг 2.9. Пример преобразования числа в строку

```
Dim i As Integer
i = 1000 * Rnd() + 1
If InStr(CStr(i), "0") > 0 Then
   MsgBox "1"
Else
   MsgBox "2"
End If
```

Другие функции преобразования типов

Кроме функций `Val` и `CStr`, В VBA имеется целый ряд функций преобразования типов данного выражения в указанный (табл. 2.4).

Таблица 2.4. *Функции преобразования типов*

Функция	Тип, в который преобразуется выражение
`CBool`	`Boolean`
`CByte`	`Byte`
`CCur`	`Currency`

***Таблица 2.4** (окончание)*

Функция	Тип, в который преобразуется выражение
CDate	Date
CDbl	Double
CDec	Decimel
CInt	Integer
CLng	Long
CSng	Single
CVar	Variant
CStr	String

Например, если имеется форма, на которой расположены два поля ввода TextBox1 и TextBox2 и требуется сложить введенные в них значения, например 2 и 5, то приводимый ниже оператор присвоит переменной s значение 25, а не 7, т. к. свойство Text поля ввода возвращает значение типа String, и поэтому реализуется конкатенация строк, а не сложение чисел.

```
s = TextBox1.Text + TextBox2.Text
```

Для того чтобы получить корректный результат, надо воспользоваться функцией преобразования типа, например CLng, которая преобразует строку в целое число типа Long, а затем сложить полученные числа, как показано в следующей инструкции.

```
s = CLng(TextBox1.Text) + CLng(TextBox2.Text)
```

Форматирование числа функцией *Format*

Чтобы представить числовое значение как дату, время, денежное значение или в специальном формате, следует использовать функцию Format, которая возвращает значение типа Variant (String), содержащее выражение, отформатированное согласно инструкциям, заданным в описании формата.

Format(*Expression*[, *Format*[, *FirstDayOfWeek* [, *FirstWeekOfYear*]]]),

где:

- *Expression* — любое допустимое выражение;
- *Format* — любое допустимое именованное или определяемое пользователем выражение формата. Примером именованного формата является Fixed — формат действительного числа с двумя значащими цифрами после десятичной точки. Примеры именованных форматов даны в табл. 2.5 и 2.6;
- *FirstDayOfWeek* — константа, определяющая первый день недели;
- *FirstWeekOfYear* — константа, определяющая первую неделю года.

Таблица 2.5. *Именованные числовые форматы*

Имя формата	Описание
`General Number`	Число без разделителя тысяч
`Currency`	Использует установки страны из панели управления. Отображает две цифры справа от десятичной точки
`Fixed`	Отображает по крайней мере одну цифру слева и две справа от десятичной точки
`Standard`	Отображает по крайней мере одну цифру слева и две справа от десятичной точки и выводит разделитель тысяч
`Percent`	Отображает число в виде процентов и выводит две цифры справа от десятичной точки
`Scientific`	Использует формат с плавающей десятичной точкой
`Yes/No`	Отображает `No`, если число равно 0 и `Yes` в противном случае
`True/False`	Отображает `False`, если число равно 0 и `True` в противном случае
`On/Off`	Отображает `Off`, если число равно 0 и `On` в противном случае

Таблица 2.6. *Именованные форматы даты и времени*

Имя формата	Описание
`General Date`	Выводит дату или время. Если нет дробной части, то выводит только дату
`Long Date`	Выводит дату в соответствии с полным форматом Windows для даты
`Medium Date`	Выводит дату в соответствии с обычным форматом Windows для даты
`Short Date`	Выводит дату в соответствии с сокращенным форматом Windows для даты
`Long Time`	Выводит часы, минуты и секунды
`Medium Time`	Выводит часы и минуты в 12-часовом формате
`Short Time`	Выводит часы и минуты в 24-часовом формате

Например, в результате действия приводимого кода из листинга 2.10 в окно **Immediate** будут выведены следующие значения:

```
General Number                4654646.544564
Currency                      4 654 646.54ð.
Fixed                         4654646.54
```

```
Standard                    4 654 646.54
Percent                     465464654.46%
Scientific                  4.65E+06
Yes/No                      Да
True/False                  Истина
On/Off                      Вкл
General Date                12.08.2002 16:15:41
Long Date                   12 Август 2002 г.
Medium Date                 12-авг-02
Short Date                  12.08.2002
Long Time                   16:15:41
Medium Time                 04:15
Short Time                  16:15
```

Листинг 2.10. Пример форматов

```
Dim x As Double
x = 4654646.544564
Debug.Print "General Number", Format(x, "General Number")
Debug.Print "Currency", , Format(x, "Currency")
Debug.Print "Fixed", , Format(x, "Fixed")
Debug.Print "Standard", , Format(x, "Standard")
Debug.Print "Percent", , Format(x, "Percent")
Debug.Print "Scientific", , Format(x, "Scientific")
Debug.Print "Yes/No", , Format(x, "Yes/No")
Debug.Print "True/False", , Format(x, "True/False")
Debug.Print "On/Off", , Format(x, "On/Off")
Debug.Print "General Date", , Format(Now, "General Date")
Debug.Print "Long Date", , Format(Now, "Long Date")
Debug.Print "Medium Date", , Format(Now, "Medium Date")
Debug.Print "Short Date", , Format(Now, "Short Date")
Debug.Print "Long Time", , Format(Now, "Long Time")
Debug.Print "Medium Time", , Format(Now, "Medium Time")
Debug.Print "Short Time", , Format(Now, "Short Time")
```

Пользовательские форматы

Пользовательские форматы позволяют настроить вид отображаемых значений так, как желает пользователь. При построении пользовательского формата применяются специальные символы, перечисленные в табл. 2.7.

Таблица 2.7. *Символы, используемые при построении пользовательского формата*

Символ	Описание
0	Резервирует позицию цифрового разряда. Отображает цифру или нуль. Если у числа, представленного параметром, есть какая-нибудь цифра в той позиции разряда, в которой в строке формата находится "0", функция отображает эту цифру параметра, если нет — в этой позиции отображается нуль
#	Резервирует позицию цифрового разряда. Отображает цифру или ничего. Если у числа, представленного параметром, есть какая-нибудь цифра в той позиции разряда, в которой в строке формата находится "#", функция отображает эту цифру параметра, если нет — в этой позиции не отображается ничего. Действие этого символа аналогично действию "0", за исключением того, что лидирующие и завершающие нули не отображаются
.	Резервирует позицию десятичного разделителя. Указание точки в строке формата определяет, сколько разрядов необходимо отображать слева и справа от десятичной точки
%	Резервирует процентное отображение числа
,	Разделитель разряда сотен от тысяч
:	Разделитель часов, минут и секунд в категории форматов Time
/	Разделитель дня, месяца и года в категории форматов Date
E+, E-, e+, e-	Разделитель мантиссы и порядка в экспоненциальном формате
d, m, y	Резервирует позицию при выводе дня, месяца, года в категории форматов Date
h, m, s	Резервирует позицию при выводе часа, минуты, секунды в категории форматов Time

В табл. 2.8 приведен ряд примеров использования пользовательских форматов.

Таблица 2.8. *Примеры пользовательских форматов*

Формат	Результат
Format(1.2 ^ 2, "##.###")	1.44
Format(1.2 ^ 2, "##.000")	1.440
Format(Sin(1) * Exp(5), "#.###e+##")	1.249e+2
Format(Now, "hh:mm:ss")	18:57:23
Format(Now, "dd/mm/yyyy")	20.01.2002

Форматирование чисел функцией *FormatNumber*

Для форматирования чисел в VBA имеется специализированная функция `FormatNumber`.

```
FormatNumber(Expression[, NumDigitsAfterDecimal [,IncludeLeadingDigit
↳[,UseParensForNegativeNumbers [,GroupDigits]]]]),
```

где:

- *Expression* — обязательный параметр, указывающий форматируемое числовое выражение;
- *NumDigitsAfterDecimal* — необязательный параметр, задающий число знаков, отображаемых после десятичной точки. Допустимыми значениями являются следующие константы: `vbTrue`, `vbFalse` и `vbUseDefault`;
- *IncludeLeadingDigit* — необязательный параметр, указывающий, надо ли отображать нулевую целую часть. Допустимыми значениями являются следующие константы: `vbTrue`, `vbFalse` и `vbUseDefault`;
- *UseParensForNegativeNumbers* — необязательный параметр, определяющий, надо ли отрицательные числа отображать в скобках. Допустимыми значениями являются следующие константы: `vbTrue`, `vbFalse` и `vbUseDefault`;
- *GroupDigits* — необязательный параметр, определяющий, надо ли группировать цифры. Допустимыми значениями являются следующие константы: `vbTrue`, `vbFalse` и `vbUseDefault`.

В табл. 2.9 приведен ряд примеров использования функции `FormatNumber`.

Таблица 2.9. *Примеры использования функции* `FormatNumber`

Формат	Результат
`FormatNumber(Sin(4), 3)`	`-0.757`
`FormatNumber(Sin(4), 3, vbTrue)`	`-0.757`
`Debug.Print (FormatNumber(Sin(4), 3, vbFalse))`	`-.757`
`FormatNumber(Sin(4), 3, vbFalse, vbTrue)`	`(.757)`
`FormatNumber(Sin(4), 3, vbFalse, vbFalse)`	`-.757`

Форматирование процентов функцией *FormatPercent*

Для форматирования процентов в VBA служит специализированная функция `FormatPercent`, которая имеет тот же самый синтаксис, что и функция `FormatNumber`. Например, следующие две инструкции

```
x = 0.2342
Debug.Print (FormatPercent(x, 2))
```

выведут в окно **Immediate** `23.42%`, т. е. число, записанное в формате процентов, у которого после десятичной точки отображаются две цифры.

Задание денежного формата функцией *FormatCurrency*

Для вывода данных в денежном формате в VBA служит специализированная функция `FormatCurrency`, которая имеет тот же самый синтаксис, что и функция `FormatNumber`. Например

`FormatCurrency(12312.3453, 2)` возвращает `12 312.35 р.`

Форматирование даты и времени функцией *FormatDateTime*

Для форматирования даты и времени в VBA имеется специализированная функция `FormatDateTime`.

`FormatDateTime(`*`Date`*`[,`*`NamedFormat`*`])`,

где:

- *`Date`* — обязательный параметр, задающий форматируемую дату;
- *`NamedFormat`* — необязательный параметр, указывающий тип форматирования. Допустимыми значениями являются следующие константы: `vbGeneralDate`, `vbLongDate`, `vbShortDate`, `vbLongTime`, `vbShortTime`.

Например, данная строка была написана 25 августа 2002 года в 18 часов 15 минут. Поэтому,

`FormatDateTime(Now, vbShortTime)` возвращает `18:15`,

`FormatDateTime(Now, vbShortDate)` возвращает `25.08.2002`.

Встроенные диалоговые окна

В проектах VBA часто встречаются две разновидности диалоговых окон: окна сообщений и окна ввода. Они встроены в VBA, и если их возможностей достаточно, то можно обойтись без проектирования диалоговых окон. Окно сообщений (`MsgBox`) выводит простейшие сообщения для пользователя, а окно ввода (`InputBox`) обеспечивает ввод информации.

Окно ввода

Функция `InputBox` выводит на экран диалоговое окно, содержащее сообщение, поле ввода и две кнопки **OK** и **Cancel.** Она устанавливает режим ожидания ввода текста пользователем и нажатия кнопки, а затем, при нажатии

на кнопку **OK**, возвращает значение типа `String`, содержащее текст, введенный в поле ввода. При нажатии кнопки **Cancel** возвращается пустая строка (`Empty`).

`InputBox(`*`Prompt`*`[, `*`Title`*`][, `*`Default`*`][, `*`Xpos`*`][, `*`Ypos`*`] [,`*`Helpfile`*`, `*`Context`*`]),`

где:

- *`Prompt`* — строковое выражение, отображаемое как сообщение в диалоговом окне. Строковое выражение *`Prompt`* может содержать несколько строк. Для разделения строк допускается использование символа возврата каретки (`Chr(13)`), символа перевода строки (`Chr(10)`) или комбинации этих символов (`Chr(13) & Chr(10)`);
- *`Title`* — строковое выражение, отображаемое в строке заголовка диалогового окна. Если этот параметр опущен, то в строку заголовка помещается имя приложения;
- *`Default`* — строковое выражение, отображаемое в поле ввода как используемое по умолчанию, если пользователь не введет другую строку. Если этот параметр опущен, то поле ввода изображается пустым;
- *`Xpos`* — числовое выражение, задающее расстояние по горизонтали между левой границей диалогового окна и левым краем экрана. Если этот параметр опущен, то диалоговое окно выравнивается по центру экрана по горизонтали;
- *`Ypos`* — числовое выражение, задающее расстояние по вертикали между верхней границей диалогового окна и верхним краем экрана. Если этот параметр опущен, то диалоговое окно помещается по вертикали примерно на одну треть высоты экрана;
- *`Helpfile`* — строковое выражение, определяющее имя файла справки, содержащего справочные сведения о данном диалоговом окне. Если этот параметр указан, то необходимо указать также параметр `Context`;
- *`Context`* — числовое выражение, определяющее номер соответствующего раздела справочной системы. Если этот параметр указан, то необходимо указать также параметр *`Helpfile`*.

Например, код из листинга 2.11 отображает на экране окно ввода, показанное на рис. 2.1.

Листинг 2.11. Окно ввода

```
Sub DemoInputBox1()
   Dim n As String
   n = InputBox("Введите ваше имя", "Пример окна ввода")
   Debug.Print n
End Sub
```

Рис. 2.1. Окно ввода

Проверка вводимых данных

При вводе данных при помощи функции `InputBox` разумно в программе предусмотреть как проверку вводимых данных, так и сам факт ввода таковых или нажатия кнопки **Cancel**. Пример из листинга 2.12 демонстрирует, как это можно сделать при вводе дат. Итак, предлагается ввести будущую дату по отношению к текущей. При помощи функции `IsDate` сначала проверяется, является ли введенное данное датой, и если таковым является, оно преобразуется функцией `CDate` к типу `Date` и сравнивается с текущей.

Листинг 2.12. Проверка вводимых значений

```
Sub DemoInputWithChecking()
   Dim d As Variant
   d = InputBox("Введите будущую дату:", "Пример", Date + 1)
   If Not IsDate(d) Then
      Select Case d
         Case ""
             MsgBox "Нажали кнопку Cancel или ничего не ввели."
         Case Else
            MsgBox "Некорректная дата."
       End Select
    Else
       Select Case CDate(d)
          Case Is <= Date
             MsgBox "Должна быть будущая дата."
          Case Is > Date
             MsgBox d & " - введенная дата."
        End Select
   End If
End Sub
```

Конечно, если вы уверены в корректности формата вводимых данных, например, таковым является строка, то проверку на тип данных осуществлять не надо, но, тем не менее, позаботиться о проверке того, была ли нажата кнопка **Cancel** или ничего не введено, стоит, например так, как это делается в листинге 2.13.

Листинг 2.13. Проверка вводимых значений

```
Sub DemoInput()
   Dim x As Variant
   x = InputBox("Введите значение:", "Пример")
   If x <> "" Then
     MsgBox x & " - введенное значение"
   Else
      MsgBox "Нажали Cancel или ничего не ввели"
   End If
End Sub
```

Ввод ссылки на диапазон

Функция `InputBox` не позволяет вводить ссылки на диапазон, но это может делать метод `InputBox` объекта `Application`. Этот метод функционально работает как одноименная функция, но у него еще имеется параметр *`Type`*, задающий тип вводимых значений. И если значение этого параметра равно 8, то как раз и вводится ссылка на диапазон. Другими возможными значениями этого параметра являются: 0 (ввод формулы), 1 (числа), 2 (строки), 4 (логического значения), 8 (ссылки на диапазон), 16 (значения ошибки) и 64 (массива значений). Например, в коде из листинга 2.14 осуществляется ввод ссылки на диапазон с проверкой, является ли действительно введенное данное диапазоном, и, если оно таковым является, то в диалоговом окне отображается его адрес.

Листинг 2.14. Ввод ссылки на диапазон

```
Sub DemoInoutRange()
   Dim r As Variant
   On Error Resume Next
   Set r = Application.InputBox("Выберете диапазон:", Type:=8)
   If IsObject(r) Then
      MsgBox r.Address
   End If
End Sub
```

Окно сообщения

Процедура `MsgBox` выводит на экран диалоговое окно, содержащее сообщение, устанавливает режим ожидания нажатия кнопки пользователем, а затем возвращает значение типа `Integer`, указывающее, какая кнопка была нажата.

`MsgBox(`*`Prompt`*`[, `*`Buttons`*`] [, `*`Title`*`] [, `*`Helpfile`*`, `*`Context`*`]),`

где:

- *`Prompt`* — строковое выражение, отображаемое как сообщение в диалоговом окне;
- *`Buttons`* — числовое выражение, представляющее сумму значений, которые указывают число и тип отображаемых кнопок, тип используемого значка, основную кнопку и модальность окна сообщения. Значение по умолчанию этого параметра равняется 0. Значения констант, определяющих число, тип кнопок и тип используемого значка приведены в табл. 2.10 — 2.12;
- *`Title`* — строковое выражение, отображаемое в строке заголовка диалогового окна. Если этот параметр опущен, то в строку заголовка помещается имя приложения;
- *`Helpfile`* — строковое выражение, определяющее имя файла справки, содержащего справочные сведения о данном диалоговом окне. Если этот параметр указан, необходимо указать также параметр *`Context`*;
- *`Context`* — числовое выражение, определяющее номер соответствующего раздела справочной системы. Если этот параметр указан, то необходимо указать также параметр *`Helpfile`*.

Таблица 2.10. *Значения параметра* `Buttons` *процедуры* `MsgBox`*, определяющие кнопки, отображаемые в диалоговом окне*

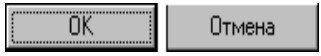
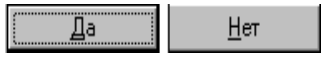
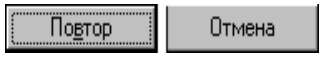
Константа	Значение	Отображаются кнопки
`vbOKOnly`	0	
`vbOKCancel`	1	
`vbAbortRetryIgnore`	2	
`vbYesNoCancel`	3	
`vbYesNo`	4	
`vbRetryCancel`	5	

Таблица 2.11. *Значения параметра* `Buttons` *процедуры* `MsgBox`*, определяющие отображаемые в диалоговом окне информационные значки*

Константа	Значение	Значок сообщения
`vbCritical`	16	
`vbQuestion`	32	
`vbExclamation`	48	
`vbInformation`	64	

Таблица 2.12. *Значения параметра* `Buttons` *процедуры* `MsgBox`*, определяющие основную кнопку в диалоговом окне*

Константа	Значение	Номер основной кнопки
`vbDefaultButton1`	0	1
`vbDefaultButton2`	256	2
`vbDefaultButton3`	512	3
`vbDefaultButton4`	768	4

При написании программ с откликом в зависимости от того, какая кнопка диалогового окна нажата, вместо возвращаемых значений удобнее использовать константы VBA, перечисленные в табл. 2.13, которые делают код программы более прозрачным для чтения и, к тому же, их легко запомнить.

Таблица 2.13. *Константы, идентифицирующие нажатую кнопку*

Константа	Значение	Нажатая кнопка
`vbOK`	1	**OK**
`vbCancel`	2	**Отмена** (`Cancel`)
`vbAbort`	3	**Прервать** (`Abort`)
`vbRetry`	4	**Повторить** (`Retry`)
`vbIgnore`	5	**Пропустить** (`Ignore`)
`vbYes`	6	**Да** (`Yes`)
`vbNo`	7	**Нет** (`No`)

Приведем демонстрационный пример (листинг 2.15) использования окон сообщений. В нем вычисляется квадрат введенного значения, и найденное значение затем последовательно отображается в пяти окнах сообщения. Первое окно является простым окном с сообщением (рис. 2.2), второе — с сообщением, выводимым в две строчки (рис. 2.3), третье — с сообщением и информационным значком (рис. 2.4), четвертое — с сообщением, информационным значком и двумя кнопками **Да** и **Нет**, причем кнопка **Да** установлена как кнопка, выполняемая по умолчанию (рис. 2.5), пятое — с сообщением, информационным значком и пользовательским заголовком (рис. 2.6).

Рис. 2.2. Простое окно сообщения

Рис. 2.3. Окно с сообщением, выводимым в две строчки

Рис. 2.4. Окно с сообщением и информационным значком

Рис. 2.5. Окно с сообщением, информационным значком и двумя кнопками

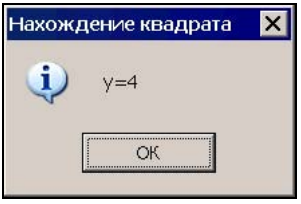


Рис. 2.6. Окно с сообщением, информационным значком и пользовательским заголовком

Листинг 2.15. Примеры окон сообщения

```
Sub DemoMsgBox()
   Dim x As Variant
   Dim y As Double
```

```
    x = InputBox("Введите x")
    y = x ^ 2
    MsgBox y
    MsgBox "x=" & x & vbCr & "y=" & y
    MsgBox "y=" & y, vbInformation
    MsgBox "y=" & y, vbInformation + vbDefaultButton1 + vbYesNo
    MsgBox "y=" & y, vbInformation, "Нахождение квадрата"
End Sub
```

Определение нажатой кнопки в окне сообщений

Процедура `MsgBox` удобна для вывода той или иной информации. Однако если надо узнать, какой выбор сделал пользователь при нажатии отображаемых в диалоговом окне кнопок, то процедуру `MsgBox` надо использовать как функцию. В этом случае значение, возвращаемое `MsgBox`, надо присваивать какой-то переменной, а ее параметры заключать в скобки. Рассмотрим пример использования функции `MsgBox` с анализом нажатой кнопки (листинг 2.16). В этом примере на экране отображается диалоговое окно с двумя кнопками **Да** и **Нет** и вопросительным значком. Клавише <Enter> назначена функция кнопки **Да**, при нажатии которой документ отображается в режиме разметки, а при нажатии кнопки **Нет** — в обычном режиме.

Листинг 2.16. Определение нажатой кнопки

```
Sub DemoYesNo()
    Dim answer As Integer
    answer = MsgBox("Отобразить документ в режите разметки", _
                 vbYesNo + vbQuestion + vbDefaultButton1, _
                "Режим отображения документа")
    Select Case answer
       Case vbYes
          ActiveWindow.View.Type = wdPrintView
       Case vbNo
          ActiveWindow.View.Type = wdNormalView
    End Select
End Sub
```

Оператор присваивания

Оператор присваивания присваивает значение выражения переменной, константе или свойству объекта. Оператор присваивания всегда включает знак

равенства (=). Например, в результате, действия следующей пары операторов:

```
x = 2
x = x + 2
```

переменной `x` будет присвоено 4.

Присваивание переменной ссылки на объект

Для присваивания переменной ссылки на объект в операторе присваивания применяется ключевое слово `Set`. В следующем примере переменной `r` присваивается ссылка на ячейку **A1**, и уже через эту переменную в ячейку **A1** вводится значение 3.

```
Dim r As Range
Set r = Range("A1")
r.Value = 3
```

В общем случае инструкция `Set` имеет следующий синтаксис.

```
Set varname = {[New] expression | Nothing}
```

где ключевое слово `New` используется при создании нового экземпляра класса, а ключевое слово `Nothing` позволяет освободить все системные ресурсы и ресурсы памяти, выделенные для объекта, на который имелась ссылка (проще говоря, она удаляет объект из памяти).

Оператор *With*

Оператор `With` избавляет программиста от утомительной обязанности использовать большое количество повторений имени одного и того же объекта при работе с его свойствами и методами. Кроме того, он структурирует код, делая его более прозрачным:

```
With Range("A1")
   .Value = 3
   .Font.Italic = True
End With
```

Операторы управления

В VBA имеется несколько операторов управления ходом выполнения программы. Функционально они делятся на две группы операторов:

- перехода и выбора (`GoTo`, `If` и `Select`);
- повтора (`For`, `For Each`, `Do Loop` и `While`).

Оператор условного перехода

Оператор условного перехода задает выполнение определенных групп инструкций в зависимости от значения выражения.

Например, если скидка (скажем, 5%) применяется только к суммам больше 1000, то в VBA это можно записать следующим образом:

```
If Сумма > 1000 Then Скидка = 0.05 Else Скидка = 0
```

или, что равносильно

```
If Сумма > 1000 Then Скидка = 0.05
```

Допускается также использование блочной формы синтаксиса, которая часто упрощает восприятие оператора условного перехода. Приводимый выше пример со скидкой можно записать в следующей эквивалентной блочной структуре:

```
If Сумма > 1000 Then
   Скидка = 0.05
Else
   Скидка = 0
End If
```

или

```
If Сумма > 1000 Then
   Скидка = 0.05
End If
```

Дерево условий может оказаться гораздо более сложным, чем просто проверка одного условия. В этом случае используется оператор `If Then ElseIf`, который позволяет проверять множественные условия. Следующий пример (листинг 2.17) демонстрирует то, как производится порядок проверки условий. В нем в зависимости от величины вводимого числа отображается сообщение о принадлежности числа: (a) интервалу [0, 1], (b) интервалу (1, 2], (c) непринадлежности числа этим двум интервалам.

Листинг 2.17. Оператор If

```
Sub DemoElseIf()
   x = InputBox("Введите число")
   If 0 <= x And x <= 1 Then
      MsgBox "Число из интервала [0, 1]"
   ElseIf 1 < x And x <= 2 Then
      MsgBox "Число из интервала (1, 2]"
   Else
      MsgBox "Число либо отрицательное, либо больше 2"
   End If
End If
```

Избегайте оператор условного перехода, если это возможно

Если это возможно, избегайте оператор условного перехода. Например, в случае кода из листинга 2.18, в котором используется оператор `If`.

Листинг 2.18. Код с оператором If

```
Sub TrueOrFalseSlower()
   Dim isYes As Boolean
   Dim i As Integer

   If i = 5 Then
      isYes = True
   Else
      isYes = False
   End If

   MsgBox isYes
End Sub
```

предпочтительно использовать код из листинга 2.19. Он не только изящнее, но и работает быстрее.

Листинг 2.19. Код без оператора If

```
Sub TrueOrFalseFaster()
   Dim isYes As Boolean
   Dim i As Integer

   isYes = (i = 5)
   MsgBox isYes
End Sub
```

Оператор выбора

Оператор выбора `Select Case` выполняет одну из нескольких групп инструкций в зависимости от значения выражения. Оператор выбора очень эффективен, когда надо проверить одну переменную, принимающую несколько значений. В следующем примере (листинг 2.20) в зависимости от величины введенного числа отображается сообщение, указывающее на величину числа или диапазон, которому оно принадлежит.

Листинг 2.20. Оператор выбора

```
Sub DemoSelect
   Dim x As Integer
   x = InputBox("Введите целое число")
   Select Case x
      Case 1
         MsgBox "Число равно 1"
      Case 2, 3
         MsgBox "Число равно 2 или 3"
      Case 4 To 6
         MsgBox "Число от 4 до 6"
      Case Is >= 7
         MsgBox "Число не менее 7"
   End Select
End Sub
```

Оператор *For Next*

Оператор `For Next` повторяет выполнение группы инструкций указанное число раз, а именно, пока переменная цикла изменяется от начального значения до конечного с указанным шагом. Если шаг не указан, то он полагается равным 1. Альтернативный способ выхода из цикла предоставляет инструкция `Exit For`. В следующем примере (листинг 2.21) при помощи оператора цикла находится сумма элементов массива.

Листинг 2.21. Оператор цикла со счетчиком цикла, имеющим единичное значение

```
Sub DemoFor1
   Dim A As Variant
   A = Array(1, 4, 12, 23, 34, 3, 23)
   s = 0
   For i = LBound(A) To UBound(A)
      s = s + A(i)
   Next
   Msgbox s
End Sub
```

В следующем коде (листинг 2.22) находится сумма всех четных целых чисел из интервала от 1 до 100.

Листинг 2.22. Оператор цикла со счетчиком цикла, имеющим значение отличное от единичного

```
Sub DemoFor2
   Dim i As Integer, s As Integer
   s = 0
   For i = 2 To 100 Step 2
      s = s + i
   Next
   Msgbox s
End Sub
```

Оператор *For Each*

Оператор `For Each` повторяет выполнение группы инструкций для каждого элемента массива или семейства. Альтернативный способ выхода из цикла предоставляет инструкция `Exit For`. Например, в следующем коде (листинг 2.23) оператор `For Each` используется для суммирования элементов массива.

Листинг 2.23. Оператор For Each

```
Sub DemoForEach
   Dim A As Variant, s Aa Double
   A = Array(1, 4, 12, 23, 34, 3, 23)
   s = 0
   For Each b In A
      s = s + b
   Next
   Msgbox s
End Sub
```

Оператор *While*

Оператор `While` выполняет последовательность инструкций, пока заданное условие возвращает значение `True`. Оператор повтора `While` в отличие от `For` выполняется не заданное число раз, а пока выполняется условие. В следующем примере (листинг 2.24) бросается игральная кость до тех пор, пока не выпадет шесть очков. При выпадении шести очков игра заканчивается, и отображается сообщение с указанием, на каком броске она закончилась.

Листинг 2.24. Оператор *While*

```
Sub DemoWhile()
   Dim attempt As Integer
   Dim score As Integer
   Randomize
   score = Int(6 * Rnd()) + 1
   attempt = 1
   While score < 6
      attempt = attempt + 1
      score = Int(6 * Rnd()) + 1
   Wend
   MsgBox "Победили на попытке: " & attempt
End Sub
```

Оператор *Do*

Оператор `Do` повторяет выполнение набора инструкций, пока условие имеет значение `True` (случай `While`) или пока оно не примет значение `True` (случай `Until`).

```
Do [{While | Until} condition]
   [statements]
   [Exit Do]
   [statements]
Loop
```

или

```
Do
   [statements]
   [Exit Do]
   [statements]
Loop [{While | Until} condition
```

В любом месте управляющей структуры `Do` может быть размещено любое число инструкций `Exit Do`, обеспечивающих альтернативные возможности выхода из цикла `Do`.

Примером использования оператора цикла `Do Until` может быть следующий код (листинг 2.25), который обеспечивает повторение цикла до тех пор, пока в поле ввода диалогового окна не будет введен пароль (в данном случае `Winnie`).

Листинг 2.25. Оператор *Do*

```
Sub DemoPassword()
   Dim ps as String
   Do
      ps = InputBox("Введите пароль")
   Loop Until ps = "Winnie"
End Sub
```

Оператор безусловного перехода *GoTo*

Оператор безусловного перехода задает переход на указанную строку внутри процедуры. Для использования оператора безусловного перехода надо какой-то строке присвоить метку. Метка должна начинаться с буквы и заканчиваться двоеточием. В качестве примера (листинг 2.26) использования оператора безусловного перехода рассмотрим игру, в которой игроку даны десять попыток броска игральной кости. В случае, если при какой-то из этих попыток выпадает шесть очков, игрок выигрывает, и игра заканчивается.

Листинг 2.26. Оператор *GoTo*

```
Sub DemoGoTo()
   Dim i As Integer
   Dim score As Integer
   Randomize
   For i = 1 To 10
      score = Int(6 * Rnd()) + 1
      If score = 6 Then GoTo lblMessage
   Next
   GoTo lblEnd
lblMessage:
   MsgBox "Выиграли при броске " & i
lblEnd:
End Sub
```

Процедура

Процедура является самостоятельной частью кода, которая имеет имя и может иметь параметры, выполнять последовательность инструкций и изменять значения своих параметров.

```
[Private | Public | Friend] [Static] Sub name [(arglist)]
   [statements]
   [Exit Sub]
   [statements]
End Sub
```

Здесь:

- `Public` — ключевое слово, указывающее на то, что процедура является *открытой* и доступна для всех других процедур во всех модулях;
- `Private` — ключевое слово, указывающее на то, что процедура является *закрытой*, и областью ее видимости является модуль;
- `Friend` — ключевое слово, используемое только в модуле класса и указывающее на то, что процедура является *дружественной*, и областью ее видимости является проект;
- `Static` — ключевое слово, указывающее на то, что локальные переменные процедуры сохраняются в промежутках времени между вызовами этой процедуры;
- *`name`* — имя процедуры, удовлетворяющее стандартным правилам именования переменных;
- *`arglist`* — список параметров, значения которых передаются в процедуру или возвращаются из процедуры при ее вызове. Разделителем в списке параметров является запятая;
- *`statements`* — любая группа инструкций, выполняемых в процедуре `Sub`;
- `Exit Sub` — оператор, приводящий к немедленному выходу из процедуры.

Вызов процедуры `Sub` из другой процедуры можно произвести несколькими способами.

- Первый способ вызова процедуры `Sub`:

  `name arglist`

  где:

  - *`name`* — имя вызываемой процедуры;
  - *`arglist`* — список фактических параметров, т. е. список параметров, передаваемых процедуре. Он должен соответствовать по количеству и типу списку параметров, заданному в процедуре при ее определении.

- Второй способ вызова процедуры `Sub` производится с помощью инструкции `Call`.

  `Call` *`name`* `(`*`arglist`*`)`

Обратите внимание, что в этом случае список фактических параметров заключается в скобки. В первом способе скобки не использовались.

Используя именованные параметры, VBA позволяет вводить фактические параметры в любом порядке и опускать необязательные (`Optional`). При этом после имени параметра ставятся двоеточие и знак равенства, после которого помещается значение параметра (фактический параметр). Приводимый ниже код (листинг 2.27) показывает основные способы передачи параметров на примере вызова процедуры `CircleLength`.

Листинг 2.27. Определение и вызов процедуры

```
Private Sub CircleLength(R As Double)
   Dim Pi As Double
   Pi = Atn(1) / 4
   Debug.Print 2 * R * Pi
End Sub

Private Sub ShowResults()
   CircleLength R:=1
   Dim Rs As Double
   Rs = 2
   CircleLength Rs
   Call CircleLength(3)
End Sub
```

Кроме процедуры `Sub` в VBA имеется процедура `Function` или просто функция. Синтаксис процедуры `Function` содержит те же элементы, что и процедура `Sub`. Инструкция `Exit Function` приводит к немедленному выходу из процедуры `Function`. Подобно процедуре `Sub`, функция является самостоятельной процедурой, которая может получать значения параметров, выполнять последовательность инструкций и изменять значения своих параметров. Однако в отличие от процедуры `Sub`, когда требуется использовать возвращаемое функцией значение, процедура `Function` может применяться в правой части выражения, как и любая другая встроенная функция, например, `Cos`. Процедура `Function` вызывается в выражении по своему имени, за которым следует список параметров, заключенный в скобки. Для возврата значения из функции следует присвоить значение имени функции. Любое число таких инструкций присваивания может находиться в любом месте функции.

В следующем коде (листинг 2.28) приводится пример функции `F`, которая находит сумму двух значений.

Листинг 2.28. Определение и вызов функции

```
Sub DemoFun()
   MsgBox F(1, 3)
End Sub

Function F(x As Double, y As Double) As Double
   F = x + y
End Function
```

Передача параметров по ссылке и значению

При вызове процедуры вы передаете в нее некоторые параметры. Внутри процедуры этим параметрам могут быть присвоены какие-то значения, которые сохранятся в них после выхода из процедуры. Таким образом, по умолчанию при передаче переменных в качестве параметров, в процедуру передаются физические адреса переменных. Поэтому внутри процедуры может быть модифицировано их содержание. Для явного указания передачи параметров в процедуру по ссылке используется ключевое слово `ByRef`. Другим способом передачи параметров в процедуру является передача их по значению. При этом способе передачи параметра в процедуру попадает не сама переменная, а ее значение. Передача параметра по значению задается ключевым словом `ByVal`. В следующем примере (листинг 2.29) показано отличие передачи параметра по ссылке от передачи параметра по значению.

Листинг 2.29. Передача параметров по ссылке и значению

```
Sub DemoByValByRef(ByVal a, b, ByRef c)
' a передается по значению
' по умолчанию b передается по ссылке
' c передается по ссылке
   a = a + 1
   b = b + a
   c = c + a
End Sub

Sub Test()
   a = 1: b = 10: c = 100
   DemoByValByRef a, b, c
   MsgBox a
```

```
' Отобразится 1
   MsgBox b
' Отобразится 12
   MsgBox c
' Отобразится 102
End Sub
```

Рекурсивные процедуры

В VBA возможно создание рекурсивных процедур, т. е. процедур, вызывающих самих себя. Классическим примером рекурсивной процедуры является процедура, возвращающая очередной член последовательности чисел Фибоначчи (листинг 2.30). Два первых члена ряда Фибоначчи равны 1, а каждый его последующий член представляет собой сумму двух предыдущих. Таким образом, *n*-е число Фибоначчи *Fi*(*n*) определяется следующим соотношением:

$$Fi(n) = Fi(n - 1) - Fi(n - 2),$$

где $Fi(1) = Fi(2) = 1$.

Листинг 2.30. Числа Фибоначчи

```
Function Fi(n As Long) As Long
   If n = 1 Or n = 2 Then
      Fi = 1
   Else
      Fi = Fi(n - 1) + Fi(n - 2)
   End If
End Function
```

Глава 3

Интегрированная среда разработки

Данная глава посвящена описанию интегрированной среды разработки приложений на VBA. Самое замечательное в этой среде то, что в ней не только удобно писать код, но и ее саму можно программировать при помощи средств VBA. Об этом также пойдет разговор в данной главе. В ней вы научитесь программно экспортировать и удалять модули, импортировать в них тестовые файлы, как удалять, так и создавать в модулях код, разрабатывать формы. Будет приведен пример программы, которая выполняется только один раз.

Где набирается код VBA

Код VBA набирается в редакторе Visual Basic. Для того чтобы попасть в этот редактор, выберите в MS Excel команду **Сервис | Макрос | Редактор Visual Basic**, или нажмите кнопку **Редактор Visual Basic**  панели инструментов **Visual Basic**, или комбинацию клавиш <Alt>+<F11>. В результате вы попадете в интегрированную среду разработки приложений (IDE) редактора Visual Basic (рис. 3.1). Возвратиться из редактора Visual Basic в рабочую книгу

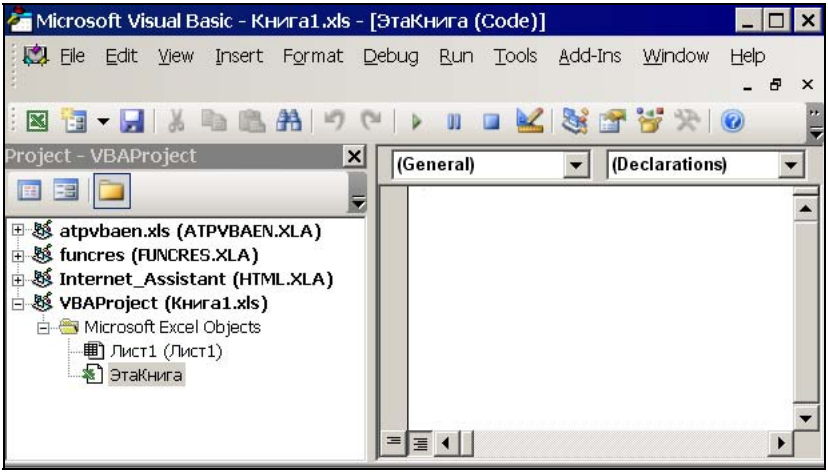


Рис. 3.1. Редактор Visual Basic

можно нажатием кнопки **View Microsoft Excel** панели инструментов **Standard**. Среда разработки приложений редактора Visual Basic имеет стандартный вид для Windows-приложений: строка меню, панель инструментов (в данном случае **Standard**) и два окна **Project — VBA Project** редактирования кода (окно **Code**).

Структура редактора VBA

Редактор VBA активизируется командой **Сервис | Макрос | Редактор Visual Basic**, или нажатием кнопки **Редактор Visual Basic** панели инструментов **Visual Basic**, или комбинацией клавиш <Alt>+<F11>.

Интерфейс редактора Visual Basic состоит из следующих основных компонентов:

- окна **Project — VBAProject**;
- окна редактирования кода (окно **Code**);
- окна форм (окно **UserForm**);
- окна свойств (окно **Properties**);
- панели инструментов (панели **Toolbox**).

Окно *Project — VBAProject*

Окно **Project — VBAProject** в редакторе Visual Basic активизируется выбором команды **View | Project Explorer** или нажатием кнопки **Project Explorer** панели инструментов **Standard**. В окне **Project — VBAProject** представлена иерархическая структура файлов форм и модулей текущего проекта (рис. 3.2).

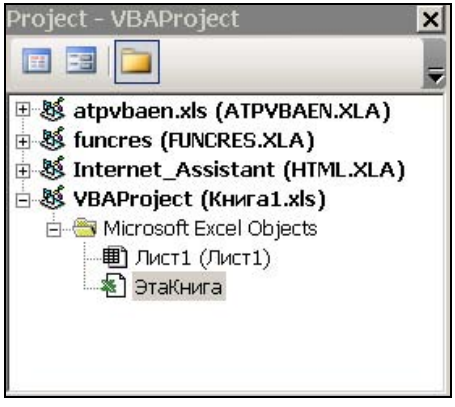


Рис. 3.2. Окно **Project — VBAProject**

В этом окне отображается реестр модулей и форм, входящих в создаваемый проект. Упрощенно говоря, *модуль* — это текстовый файл, в котором набирается код. Двойным щелчком на значке модуля в окне **Project — VBAProject** можно открыть соответствующий модуль. Значок активного модуля в окне **Project — VBAProject** выделяется серым цветом.

В проекте автоматически создается по модулю для каждого рабочего листа и для всей книги. Кроме того, модули создаются для каждой пользовательской формы, макросов и классов. По своему предназначению модули делятся на два типа: модули объектов и стандартные. К стандартным модулям относятся те, на которых записываются макросы. Такие модули добавляются в проект выбором команды **Insert | Module**. К модулям объектов относятся модули, связанные с рабочей книгой, рабочими листами, формами, а также модули класса.

Формы создаются выбором команды **Insert | UserForm**, а модули класса — **Insert | Class Module**. По мере создания, добавления и удаления файлов из проекта эти изменения отображаются в окне проекта. Отметим, что удаление файла из окна проекта производится выбором значка файла с последующим выбором команды **File | Remove**.

Копирование модулей и форм из одного проекта в другой

В окне проекта выводятся проекты всех открытых рабочих книг. Это позволяет легко копировать формы и модули из одного проекта в другой при помощи простой буксировки значка файла в окне **Project — VBAProject**.

Окно редактирования кода

Двойной щелчок на значке элемента проекта в окне проекта открывает окно редактора кода для соответствующего модуля (рис. 3.3). Открыть модуль в редакторе кода для соответствующего объекта (например, рабочего листа) можно также выбором значка этого объекта в окне проекта с последующим выбором команды **View | Code**. Вернуться назад из модуля к соответствующему объекту можно выбором команды **View | Object**.

Окно редактирования кода служит в качестве редактора для ввода кода процедур приложения. Код внутри модуля организован в отдельные разделы для каждого объекта, программируемого в модуле. В окне редактирования доступны два режима представления кода: просмотр отдельной процедуры и всего модуля. Переключение режимов работы окна редактирования кода осуществляется выбором одной из двух кнопок в нижнем левом углу окна редактирования кода (левая — отдельная процедура, правая — все процедуры модуля), либо установкой или снятием флажка **Default to Full Module View** вкладки **Editor** диалогового окна **Options**, отображаемого на экране

командой **Tools | Options**. Если установлен режим просмотра всех процедур модуля, то процедуры можно отображать с разделителями (горизонтальной чертой, разделяющей две соседние процедуры) или без них. Отображением или неотображением разделителей управляет флажок **Procedure Separator**.

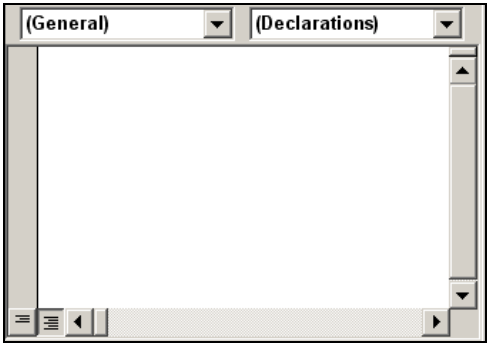


Рис. 3.3. Окно редактирования кода

Два раскрывающихся списка в верхней части окна редактора кода облегчают ориентацию в процедурах. Левый раскрывающийся список позволяет выбрать объект, а правый содержит список событий, допустимых для выбранного в левом списке объекта. Кроме того, выбор объекта и события приводит к созданию в редакторе кода первой и последней инструкции процедуры обработки события, связанного с выбранным объектом, если такой еще не существовало.

Совет

Всегда используйте раскрывающиеся списки окна редактора кода для создания первой и последней инструкции процедуры обработки события, связанного с выбранным объектом. Использование автоматических средств, с одной стороны, убыстряет процесс создания кода (не надо его набирать вручную), а с другой стороны, позволяет избежать опечаток. О других средствах редактора кода, облегчающих набор кода, поговорим в следующем разделе.

Интеллектуальные возможности редактора кода

Написание программ существенно облегчается за счет способности редактора кода автоматически завершать написание операторов, свойств и параметров. При написании кода редактор сам предлагает пользователю список компонентов, логически завершающих вводимую пользователем инструкцию. Например, при наборе кода:

```
Range("A1").
```

после ввода точки на экране отобразится список компонентов (рис. 3.4), которые логически завершают данную инструкцию. Двойной щелчок на выбран-

ном элементе из этого списка или нажатие клавиши <Tab> вставляет выбранное имя в код программы. При этом использование клавиши <Tab> вместо мыши выглядит предпочтительней, так как эта клавиша находится прямо под рукой, и нажатие на нее производится только одним движением пальца левой руки, что не требует ни времени, ни усилий. В то же время работа с мышью, в данном случае, выглядит чрезмерно утомительным делом.

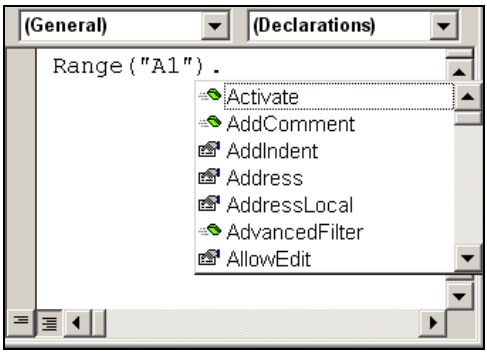


Рис. 3.4. Список компонентов

Автоматическое отображение списка компонентов происходит только при установленном флажке **Auto List Members** вкладки **Editor** диалогового окна **Options**, отображаемого на экране выбором команды **Tools | Options**.

Список компонентов можно выводить на экран нажатием комбинации клавиш <Ctrl>+<J>, при этом он отображается как при установленном, так и снятом флажке **Auto List Members** вкладки **Editor** диалогового окна **Options**.

Совет

Список компонентов отображается только для существующих в форме или на рабочем листе элементов управлений. Поэтому, если в ваш проект входят элементы управления, сначала создайте их, а уж потом набирайте код.

Отображение списка компонентов, логически завершающих вводимую инструкцию, является одним из интеллектуальных качеств редактора кода. Этим качеством интеллектуальные ресурсы редактора кода не исчерпываются. Другим таким его качеством является автоматическое отображение на экране сведений о процедурах, функциях, свойствах и методах после набора их имени (рис. 3.5).

Автоматическое отображение на экране сведений о процедурах, функциях, свойствах и методах после ввода их имени происходит только при установленном флажке **Auto Data Tips** вкладки **Editor** диалогового окна **Options**, отображаемого на экране выбором команды **Tools | Options**.

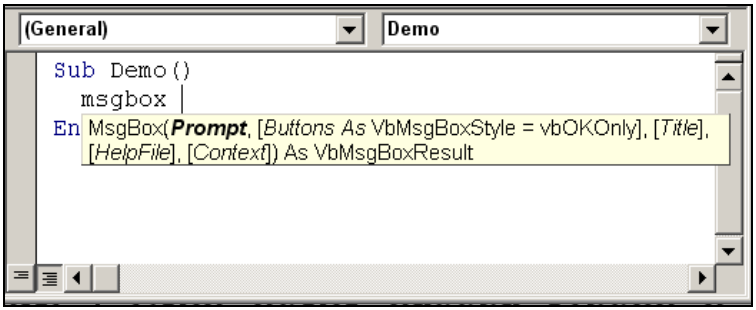


Рис. 3.5. Отображаемые сведения о вводимой процедуре

Описанную выше всплывающую подсказку можно также выводить на экран нажатием комбинации клавиш <Ctrl>+<I>. При этом всплывающая подсказка отображается как при установленном, так и снятом флажке **Auto Data Tips** вкладки **Editor** диалогового окна **Options**.

Редактор кода также производит автоматическую проверку синтаксиса набранной строки кода сразу после нажатия клавиши <Enter>. Если после набора строки и нажатия клавиши <Enter> строка выделяется красным цветом, то это указывает на наличие синтаксической ошибки в набранной строке. Эту ошибку необходимо найти и исправить. Кроме того, если установлен флажок **Auto Syntax Check** вкладки **Editor** диалогового окна **Options**, помимо выделения красным цветом фрагмента кода с синтаксической ошибкой, на экране отображается диалоговое окно, поясняющее, какая, возможно, произошла ошибка.

Редактор кода обладает еще одной мощной интеллектуальной возможностью, увеличивающей эффективность работы пользователя. Если расположить курсор на ключевом слове языка VBA или имени процедуры, функции, свойства или метода и нажать клавишу <F1>, то на экране появится окно со справкой об этой функции. Обычно к этой справке прилагается пример использования кода. А, имея под рукой пример, всегда легче разобраться в ситуации, которая, возможно, при написании программы возникла в том месте, где вам потребовалось обратиться к справочному средству.

Окно *UserForm* (Редактирование форм)

Для создания диалоговых окон разрабатываемых приложений в VBA используются формы. Редактор форм является одним из основных инструментов визуального программирования. Форма в проект добавляется выбором команды **Insert | UserForm**. В результате на экран выводится незаполненная форма с панелью инструментов **Toolbox** (рис. 3.6).

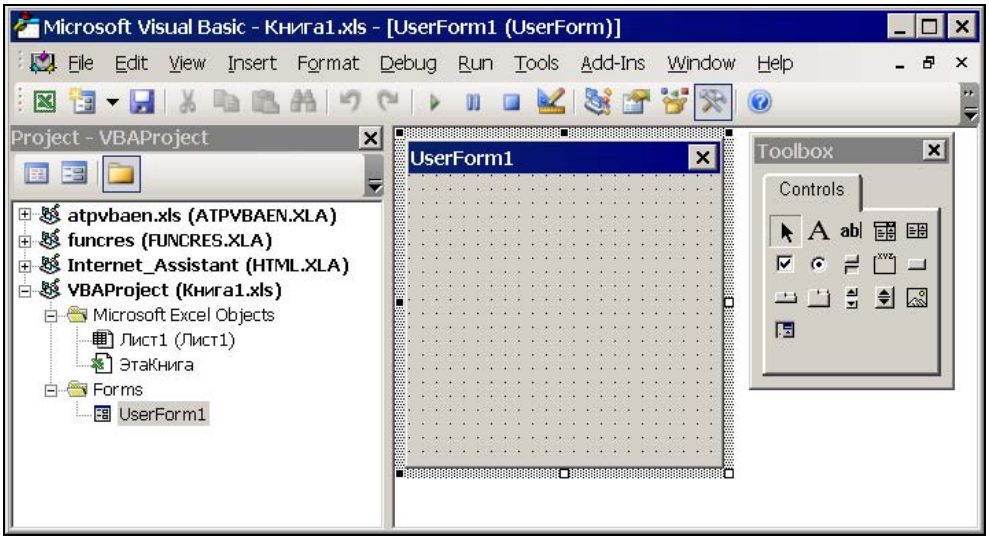


Рис. 3.6. Окно редактирования форм и панель инструментов **Toolbox**

Используя панель инструментов **Toolbox** из незаполненной формы, можно сконструировать любое требуемое для приложения диалоговое окно. Размещение нового управляющего элемента на форме осуществляется следующей последовательностью действий:

1. Щелкните на значке того элемента, который собираетесь разместить на форме.
2. Поместите указатель мыши на то место, где будет располагаться управляющий элемент.
3. Нажмите левую кнопку мыши и, не отпуская ее, растяните появившийся прямоугольник до требуемых размеров.
4. Отпустите кнопку мыши. Элемент управления создан на искомом месте.

Размеры формы и расположенных на ней элементов управления можно изменять. Технология изменения их размеров стандартная для Windows: необходимо выделить изменяемый элемент, поместить указатель мыши на один из размерных маркеров и протащить его при нажатой левой кнопке мыши так, чтобы объект принял требуемые размеры. Окно редактирования форм поддерживает операции буфера обмена. Таким образом, можно копировать, вырезать и вставлять элементы управления, расположенные на поверхности формы.

Для облегчения размещения и выравнивания элементов управления используется сетка. Параметры сетки устанавливаются в группе **Form Grid Settings** вкладки **General** диалогового окна **Options** вызываемого командой **Tools** | **Options**:

- ❒ флажок **Show Grid** управляет отображением сетки на форме;

- поля **Width** и **Height** устанавливают расстояние по горизонтали и вертикали между соседними узлами сетки;
- флажок **Align Controls to Grid** управляет привязыванием элементов управления к сетке.

Иногда требуется переместить группу элементов управления в одном и том же направлении. Например, если несколько кнопок находятся на одной линии и их надо переместить, не нарушая строя, на несколько шагов сетки. При работе с группой элементов управления как с одним объектом необходимо первоначально эту группу сформировать. Элементы управления можно объединять в группу, выделяя элементы по одному щелчками мыши по ним при удерживаемой клавише <Ctrl>. Для отмены выделения группы достаточно щелкнуть в любом месте формы, не занятом элементами управления.

После формирования группы элементов управления можно легко перемещать их совместно и изменять размеры при помощи команд выпадающего меню **Format**, которые перечислены в табл. 3.1.

Таблица 3.1. *Команды выпадающего меню* ***Format***

Команда	Описание
Align | Lefts (Centers, Rights)	Выровнять группу элементов по левому краю (центру, правому краю)
Align | Tops (Middles, Bottoms)	Выровнять группу элементов по верхнему краю (середине, нижнему краю)
Align | To Grids	Выровнять по решетке
Make Same Size | Width (Height, Both)	Сделать элементы управления одной и той же высоты (ширины; как высоты, так и ширины)
Size to Fit	Установить размеры элементов управления так, чтобы они совпадали с размерами отображаемых в них надписей, рисунков
Size to Grid	Установить размеры элементов управления по сетке
Horizontal Spacing | Make Equal (Increase, Decrease, Remove)	Установить равное по горизонтали расстояние между элементами управления (увеличить его, уменьшить, удалить)
Vertical Spacing | Make Equal (Increase, Decrease, Remove)	Установить равное по вертикали расстояние между элементами управления (увеличить его, уменьшить, удалить)
Center in Form | Horizontally (Vertically)	Центрирование элемента управления в форме по горизонтали (вертикали)
Arrange Buttons | Bottom (Right)	Расположение кнопок вдоль основания (правого края формы)

Таблица 3.1 *(окончание)*

Команда	Описание
Group	Группировка элементов управления
Ungroup	Разгруппировка группы элементов управления
Order | Bring to Front (**Sent to Back**)	Расположить элементы управления на переднем (заднем) плане

Иногда на форме могут быть созданы лишние элементы управления. Для удаления лишних элементов управления достаточно их выделить и нажать клавишу <Del>.

Окно *Properties* (Свойства)

В окне свойств перечисляются основные установки свойств выбранной формы или элемента управления. Используя это окно можно просматривать свойства и изменять их установки. Для просмотра свойств выбранного объекта надо либо щелкнуть кнопку **Properties Window**  , либо выбрать команду **View | Properties Window**, либо нажать клавишу <F4> (рис. 3.7).

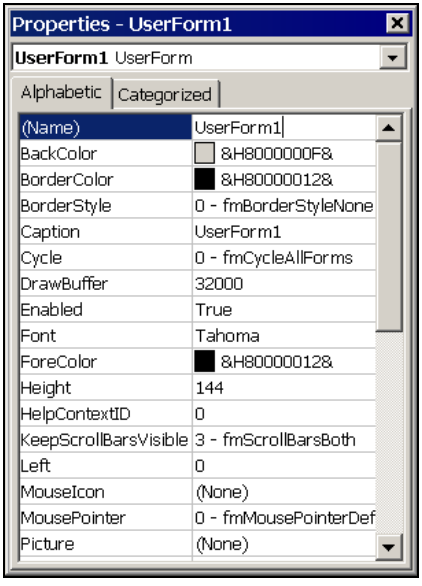


Рис. 3.7. Окно **Properties**

Окно свойств состоит из двух составных частей: верхней и рабочей. В верхней части окна свойств располагается раскрывающийся список, из которого

можно выбрать любой элемент управления текущей формы или саму форму. Рабочая часть окна свойств состоит из двух вкладок: **Alphabetic** и **Categorized**, отображающих набор свойств в алфавитном порядке или по категориям. На обеих вкладках свойство `Name` (имя элемента управления) идет первым. Изменяются значения свойств одним из следующих способов:

- вводом с клавиатуры значения свойства в соответствующее поле;
- выбором из раскрывающегося списка. Так можно выбрать значения большинства свойств. Раскрывающийся список активизируется щелчком на соответствующем поле окна свойств.

Совет

Свойство `Name` часто используется при создании процедур обработки событий для элемента управления. Оно определяет его имя. Присвоение объектам значимых имен облегчает чтение кода, делает его прозрачным. Например, предположим, что на форме создана кнопка `CommandButton1`, при нажатии которой закрывается форма. Тогда, следующий код:

```
Private Sub cmdExit_Click()

End Sub
```

значительно более информативен, чем код:

```
Private Sub CommandButton1_Click()

End Sub
```

Окно *Object Browser* (Просмотр объектов)

Окно **Object Browser** отображается на экране выбором команды **View | Object Browser**, или нажатием кнопки **Object Browser**  , или нажатием кнопки <F2> (рис. 3.8). В этом окне приведен список всех объектов, которые имеются в системе и которые можно использовать при создании проекта.

Окно **Object Browser** состоит из трех основных частей:

- раскрывающегося списка **Project/Library** в левом верхнем углу окна. В этом раскрывающемся списке можно выбрать различные проекты и библиотеки объектов. В частности, библиотеки объектов Excel, VBA, Office и VBAProject (объекты пользовательского проекта). Выбор в списке элемента <**All Libraries**> отображает список объектов всех библиотек;
- списка **Classes**. После выбора из раскрывающегося списка **Project/Library** просматриваемой библиотеки, например VBA, все классы объектов выбранной библиотеки выводятся в списке **Classes**;
- списка **Members**. После выбора класса из списка **Classes** просматриваемой библиотеки, например `FileSystem`, все компоненты выбранного

класса выводятся в списке **Members**. При выделении строки в списке **Members** в нижней части окна **Object Browser** приводится дополнительная информация о выбранном компоненте. Кроме того, если нажать на кнопку **Help**, расположенную на панели инструментов в правой верхней части окна **Object Browser**, то на экране отобразится окно **Справочник Visual Basic** с подробной справкой по выделенному компоненту.

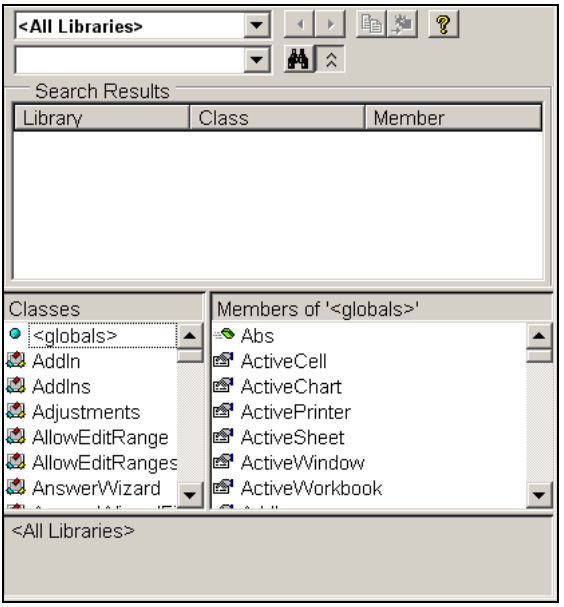


Рис. 3.8. Окно **Object Browser**

Программирование среды разработки

Среда разработки не только обеспечивает создание кода, но и ее саму можно программировать средствами VBA. Для того чтобы это можно было делать, надо сначала получить право воздействия на нее со стороны кода. Для этого надо выбрать команду **Сервис | Макрос | Безопасность**. На вкладке **Надежные издатели** окна **Безопасность** установите флажок **Доверять доступ к Visual Basic Project** и нажмите кнопку **OK**.

Данные о среде разработке инкапсулируются в объекте `VBE`, который в себе, как в контейнере, содержит следующие семейства объектов:

- семейство `VBProjects` реализует весь набор проектов;
- семейство `AddIns` предоставляет доступ ко всем настройкам;
- семейство `Windows` инкапсулирует данные обо всех окнах;

- семейство `CodePanes` предоставляет доступ к модулям с кодом;
- семейство `CommandBars` реализует набор панелей инструментов.

Для того чтобы использовать эти объекты в проекте, надо установить ссылку на библиотеку Microsoft Visual Basic for Applications Extensibility. Для этого:

- выберите команду **Tools | References**;
- в списке **Available References** окна **References** выберите элемент **Microsoft Visual Basic for Applications Extensibility** и нажмите кнопку **OK**.

Экспортирование модуля

Код из листинга 3.1 добавляет в другую рабочую книгу с именем **Book2** стандартный модуль и импортирует в него содержание модуля **Module2** данной рабочей книги. Здесь используется объект `CodeModule`, инкапсулирующий в себе код, набранный в данной компоненте. Свойство `Lines` возвращает указанное число строк кода, начиная со специфицированной линии, а свойство `CountOfLines` — общее число строк. Метод `Add` производит вставку в среду разработки новой компоненты, тип которой задается его параметром, который может иметь следующие допустимые значения: `vbext_ct_StdModule` или 1 (стандартный модуль), `vbext_ct_ClassModule` или 2 (модуль класса), `vbext_ct_MSForm` или 3 (форма), `vbext_ct_ActiveXDesigner` или 11 (ActiveX дизайнер) и `vbext_ct_Document` или 100 (модуль документа). Метод `AddFromString` вставляет в модуль строку текста.

Листинг 3.1. Экспортирование модуля

```
Sub ExportModule()
   Dim wb As Workbook
   Dim CodeLines As String
   With ThisWorkbook.VBProject.VBComponents("Module2").CodeModule
      CodeLines = .Lines(1, .CountOfLines)
   End With
   Set wb = Workbooks("Book2")
   With wb.VBProject.VBComponents
      .Add vbext_ct_StdModule
      .Item(.Count).CodeModule.AddFromString CodeLines
   End With
End Sub
```

Удаление модуля

Модуль можно не только создавать и импортировать в него данные, но и удалять с помощью метода `Remove`. Следующий код (листинг 3.2) демонстрирует, как можно удалить модуль из рабочей книги, в которой выполняется данный макрос.

Листинг 3.2. Удаление модуля

```
Sub DeleteModule(moduleName As String)
   Dim vbc As VBComponent
   Set vbc = ThisWorkbook.VBProject.VBComponents(moduleName)
   ThisWorkbook.VBProject.VBComponents.Remove vbc
   Set vbc = Nothing
End Sub
```

Программа, которая выполняется только один раз

Используя метод `Remove` легко создать программу, которая выполняется только один раз, после чего уничтожает тот модуль, в котором она расположена. Например, добавьте в проект стандартный модуль, переименуйте его в **TmpModule** и наберите в нем следующий код (листинг 3.3). После выполнения этой программы модуль **TmpModule** будет удален из проекта.

Листинг 3.3. Программа, которая выполняется только один раз

```
Sub DeleteAfterRun()
   MsgBox "Hello, World!"
   Dim vbc As VBComponent
   Set vbc = ThisWorkbook.VBProject.VBComponents("TmpModule")
   ThisWorkbook.VBProject.VBComponents.Remove vbc
   Set vbc = Nothing
End Sub
```

Импорт данных в модуль из текстового файла

Данные можно импортировать не только из другой книги, но и из текстового файла. В этом случае надо воспользоваться методом `AddFromFile`. Следующий код (листинг 3.4) демонстрирует подобный подход, при этом код импортируется из файла с указанным именем и вставляется в начало специфицированного модуля рабочей книги, в которой расположен данный макрос.

Листинг 3.4. Импорт данных в модуль из текстового файла

```
Sub ImportFromFile(moduleName As String, fileName As String)
   Dim cm As CodeModule
   Set cm = ThisWorkbook.VBProject.VBComponents(moduleName).CodeModule
   cm.AddFromFile fileName
   Set cm = Nothing
End Sub
```

Очистка содержания модуля

Для очистки модуля от его содержания достаточно удалить из него все строки, что можно реализовать методом `DeleteLines` объекта `CodeModule`, как это делается в листинге 3.5. У этого метода имеются два параметра. Первый указывает на строку, с которой надо производить удаление, а второй — на количество удаляемых строк, которое в нашем случае должно совпадать с общим числом строк в модуле, а это число возвращается свойством `CountOfLines`.

Листинг 3.5. Очистка содержания модуля

```
Sub ClearModule(ByVal wb As Workbook, _
                ByVal ClearModuleName As String)
   With wb.VBProject.VBComponents(ClearModuleName).CodeModule
      .DeleteLines 1, .CountOfLines
   End With
End Sub
```

Как проверить, существует ли модуль либо процедура

Для того чтобы проверить, существует ли модуль либо процедура, можно воспользоваться функциями, определенными в листинге 3.6. Они определяют длину имени специфицированного модуля или процедуры возвращаемого свойством `Name`. Если эта длина больше нуля, то искомый элемент существует. Конечно, в коде надо предусмотреть перехват возможной ошибки, генерируемой при ссылке на несуществующий элемент.

Листинг 3.6. Проверка существования модуля либо процедуры

```
Function ModoleExists(ModuleName As String, wb As Workbook) As Boolean
   On Error Resume Next
   ModuleExists = Len(wb.VBProject.VBComponents(ModuleName).Name) <> 0
```

```
End Function

Function ProcedureExists(ProcedureName As String, _
                         ModuleName As String, _
                         wb As Workbook) As Boolean
   On Error Resume Next
   If ModuleExists(ModuleName) Then
      ProcedureExists = wb.VBProject.VBComponents(ModuleName) _
         .CodeModule.ProcStartLine(ProcedureName, vbext_pk_Proc) <> 0
   End If
End Function
```

Автоматическое создание кода в модуле

Код можно не только импортировать из других источников, но и автоматически его создавать. Описанная в листинге 3.7 процедура `InsertProcedure` как раз и вставляет в указанный модуль процедуру `Hello`, отображающую стандартное диалоговое окно с приветствием `Hello World!` У процедуры `InsertProcedure` имеются два параметра. Первый задает ссылку на рабочую книгу, а второй — на модуль, в который будет вставляться код. Вставка строки кода реализуется методом `InsertLines`, первый параметр которого задает номер строки, а второй — вводимый текст.

Листинг 3.7. Автоматическое создание кода в модуле

```
Sub InsertProcedure(ByVal wb As Workbook, _
                    ByVal ModuleName As String)
   Dim cm As CodeModule
   Dim idx As Long
   Set cm = wb.VBProject.VBComponents(ModuleName).CodeModule
   With cm
     idx = .CountOfLines + 1
     .InsertLines idx, "Sub Hello()"
     idx = idx + 1
     .InsertLines idx, _
        "    Msgbox ""Hello World!"",vbInformation"
     idx = idx + 1
     .InsertLines idx, "End Sub"
      End With
   Set cm = Nothing
```

```
End Sub

Sub Demo()
   InsertProcedure ActiveWorkbook, "Module1"
End Sub
```

Удаление кода процедуры из модуля

Процедуры можно не только создавать, но и удалять. Описанная в листинге 3.8 процедура `DeleteProcedure` как раз и демонстрирует, как это можно делать. У нее имеются три параметра. Первый и второй задают ссылки на рабочую книгу и модуль, в которых находится процедура, а третий — на саму процедуру. Для того чтобы удалить указанную процедуру, надо воспользоваться свойствами `ProcStartLine` и `ProcCountLines`. Эти свойства определяют номер данной процедуры в модуле и число занимаемых ей строк. Они могут возвращать аналогичные данные для свойств, что определяется вторым параметром, у которого имеются следующие допустимые значения: `vbext_pk_Get`, `vbext_pk_Let`, `vbext_pk_Set` и `vbext_pk_Proc`. Метод `DeleteLines` удаляет из модуля указанное число строк, начиная с данной.

Листинг 3.8. Удаление кода процедуры из модуля

```
Sub DeleteProcedure(ByVal wb As Workbook, _
      ByVal ModuleName As String, _
      ByVal ProcedureName As String)
   Dim cm As CodeModule
   Dim Start As Long, Count As Long
   Set cm = wb.VBProject.VBComponents(ModuleName).CodeModule
   Start = cm.ProcStartLine(ProcedureName, vbext_pk_Proc)
   If Start > 0 Then
      Count = cm.ProcCountLines(ProcedureName, vbext_pk_Proc)
      cm.DeleteLines Start, Count
   End If
   Set cm = Nothing
End Sub

Sub DemoDelete()
   DeleteProcedureCode ActiveWorkbook, "Module1", "Hello"
End Sub
```

Список всех процедур модуля

В данном разделе приводится пример кода (листинг 3.9), который выводит список всех процедур, размещенных в указанном модуле (в данном случае **Module1**). Процедура работает по следующей схеме. Сначала определяется, сколько строк кода имеется в области объявления, при помощи свойства `CountOfDeclarationLines` объекта `CodeModule`. А затем, сколько строк кода имеется в очередной из процедур, свойством `ProcCountLines`. Свойство `ProcOfLine` возвращает имя процедуры, объявленной на специфицированной строке.

Листинг 3.9. Список всех процедур модуля

```
Sub ProceduresList()
   Dim cm As CodeModule
   Dim n As Long
   Dim Msg As String
   Dim ProcName As String
   Set cm = ThisWorkbook.VBProject.VBComponents("Module1").CodeModule
   n = cm.CountOfDeclarationLines + 1
   Do Until n >= cm.CountOfLines
      Msg = Msg & cm.ProcOfLine(n, vbext_pk_Proc) & vbCr
      n = n + cm.ProcCountLines(cm.ProcOfLine(n, vbext_pk_Proc), _
                                vbext_pk_Proc)
   Loop
   MsgBox Msg
End Sub
```

Создание пользовательской формы

В среде разработки можно автоматизировать не только процесс ввода кода, но и создание форм. Как это делается, демонстрируется в примере из листинга 3.10. В нем конструируется форма, на которой расположена надпись. Форма создается методом `Add` семейства `VBComponents`, у которого значение параметра установлено равным `vbext_ct_MSForm`, а надпись создается методом `Add` семейства `Controls`.

Листинг 3.10. Создание пользовательской формы

```
Sub FormBuilder()
   Dim myForm As VBComponent
   Dim myLabel As Control
```

```
    Set myForm = _
       ActiveWorkbook.VBProject.VBComponents.Add(vbext_ct_MSForm)
    With myForm
       .Properties("Height") = 50
       .Properties("Width") = 100
       .Name = "UserForm1"
       .Properties("Caption") = "Hello!"
    End With
    Set myLabel = myForm.Designer.Controls.Add("Forms.Label.1")
    With myLabel
       .Name = "Label1"
       .Caption = "Hello, World!"
       .Left = 10
       .Top = 10
       .Height = 20
       .Width = 80
    End With
End Sub
```

Построение формы с элементом управления, реагирующим на события

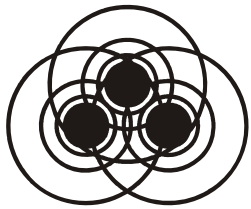
Форму можно создавать, конечно, не только с метками, но и любыми другими элементами управления — кнопками, переключателями и т. д. Технология создания таких форм такая же, как и в предыдущем разделе, но имеется одна тонкость — все эти элементы управления в отличие от надписи должны уметь реагировать на события. Как их научить этому? Да очень просто! В код надо добавить инструкции, которые и создают процедуру обработки соответствующего события, как мы это уже делали в разделе *Автоматическое создание кода в модуле*. В следующем примере (листинг 3.11) в форму добавляется кнопка **OK**, нажатие на которую приведет к отображению диалогового окна с классическим приветствием Hello, World! Для выполнения этого кода кроме ссылки на библиотеку **Microsoft Visual Basic for Applications Extensibility** надо также установить ссылку на библиотеку **Microsoft Form 2.0 Object Library**. Кроме того, обратите внимание на следующую особенность этого кода — после закрытия окна форма автоматически удаляется из редактора Visual Basic.

Листинг 3.11. Построение формы с элементом управления, реагирующим на события

```
Sub FormBuilder()
    Dim frm As VBComponent
    Dim btn As MSForms.CommandButton
```

```
    Dim NumberOfLine As Integer
    Dim prjt As Variant
    On Error Resume Next

    If Err <> 0 Then
       MsgBox "Ошибка при доступе к среде", vbCritical
       On Error GoTo 0
       Exit Sub
    End If

    Application.VBE.MainWindow.Visible = False

    Set frm = ThisWorkbook.VBProject.VBComponents.Add(vbext_ct_MSForm)
    With frm
       .Properties("Caption") = "Пример"
       .Properties("Width") = 100
       .Properties("Height") = 100
    End With

    Set btn = frm.Designer.Controls.Add("Forms.CommandButton.1")
    With btn
       .Caption = "OK"
       .Left = 10
       .Top = 30
       .Width = 40
       .Height = 20
       .Name = "btnOK"
    End With

    With frm.CodeModule
       NumberOfLine = .CountOfLines
       .InsertLines NumberOfLine + 1, "Sub btnOK_Click()"
       .InsertLines NumberOfLine + 2, "MsgBox ""Hello, World!"""
       .InsertLines NumberOfLine + 3, "End Sub"
    End With
    VBA.UserForms.Add(frm.Name).Show
    ThisWorkbook.VBProject.VBComponents.Remove frm
End Sub
```

Глава 4



Элементы ООП в VBA

В VBA, наряду с огромным числом встроенных объектов (`Application`, `Workbook`, `Worksheet`, `Range` и т. д.), предусмотрена возможность создания пользовательских объектов. Использование пользовательских объектов позволяет сократить текст программ, делать их более прозрачными и понятными. Пользовательские объекты в VBA обладают свойствами, полями, методами, могут слушать события, но, к сожалению, не обладают привычным для объектно-ориентированных языков механизмом наследования.

Классы и их экземпляры

Объекты являются представителями или экземплярами классов. Образно говоря, классы являются формами, из которых пекутся конкретные объекты. Метод, активизируемый объектом в ответ на сообщение, определяется классом, к которому принадлежит получатель сообщения. Все объекты одного и того же класса используют одинаковые методы в ответ на одинаковые сообщения. Членами класса являются поля, свойства, методы и события. Рассмотрим их подробнее:

- *полем* называется переменная, содержащая некоторое значение. Таким образом, поле предоставляет информацию об объекте. Например, если имеется объект `Car` (автомобиль), то его полем может быть поле `Cylinders` (цилиндры), хранящее информацию о числе цилиндров в двигателе автомобиля;
- *метод* — это код, программирующий некоторые действия, которые должен совершить объект. Например, объект `Car` может быть перекрашен методом `Repaint`;
- *свойство* играет ту же роль, что и поле, а именно, предоставляет информацию об объекте, но при его создании используются специальные процедуры `Property`, которые предоставляют больше возможностей как по установке значения, так и его возвращения. Свойство позволяет отделить данные от объекта, сделав их более устойчивыми;

- *событие* позволяет объектам оповещать друг друга о произошедшем изменении в ситуации. События часто используются в графических интерфейсах для оповещения, например, о нажатии пользователем кнопки, но они также хорошо подходят и для любого другого вида оповещений, например, о получении электронной почты.

Объявление класса

Классы конструируются в модулях классов, которые создаются выбором команды **Insert | Class Module**. При создании класса надо предусмотреть его инициализацию, описание свойств и методов, которыми будет наделен объект.

Опишем процесс создания класса в виде следующей последовательности шагов:

1. Выберите команду **Insert | Class Module.** Откроется окно нового модуля класса.
2. Нажмите клавишу <F4> и в появившемся окне **Properties** установите значение свойства `Name` равным имени класса. Имя модуля класса совпадает с именем класса.
3. Класс инициализируется при помощи необязательной для объявления процедуры `Class_Initialize`. В ней можно задать значения, назначаемые полям при конструировании экземпляра класса.
4. Допустимо также объявление процедуры `Class_Terminate` для описания процесса удаления объекта из памяти по завершению работы с ним.

В качестве примера создадим класс `Client`, моделирующий информацию о клиенте банка (листинг 4.1). В этом классе имеются только два поля `Name` и `Account`, которые задают имя клиента и размер его счета. Модификатор доступа `Public` устанавливает, что поля доступны для всех внешних кодов и могут быть опрошены и изменены любым из них.

Листинг 4.1. Модуль класса `Client`

```
Public Name As String
Public Account As Long
```

Создание экземпляра класса

Для того чтобы использовать созданный класс, нужно иметь возможность получения экземпляра этого класса. *Экземпляр* — это объект, имеющий тип данного класса. Для любого созданного класса можно получить экземпляры так же, как для любого другого типа данных, но при объявлении объекта

надо указать ключевое слово New. После создания экземпляра класса у него можно считывать или устанавливать значения полей, свойств, применять к нему методы. Например, в следующем коде (листинг 4.2) создается экземпляр класса Client, у которого устанавливаются значения полей Name и Account. Кроме того, некоторая переменная может быть сначала объявлена как объектная (в данном случае типа Client), а затем при помощи оператора присваивания Set ей уже может быть назначено значение.

Листинг 4.2. Создание экземпляра класса. Стандартный модуль

```
Sub TestClient()
   Dim c1 As New Client
   c1.Name = "Bond"
   c1.Account = 300000
   Dim c2 As Client
   Set c2 = New Client
   c2.Name = "Pooh"
   c2.Account = 200000
   Debug.Print c1.Name & vbTab & c1.Account
   Debug.Print c2.Name & vbTab & c2.Account
End Sub
```

Инициализация значений полей

Начальные значения полей экземпляра класса можно устанавливать в зависимости от бизнес-логики проекта. Для этого достаточно в процедуре обработки события Initialize модуля класса полям присвоить требуемые значения. Например, следующая модификация класса Client (листинг 4.3, *а*) обеспечит то, что все создаваемые экземпляры этого класса первоначально имеют фамилию Ivanov с нулем денег на счету. С классом также связано событие Terminate, которое генерируется, когда экземпляр класса удаляется из памяти путем установки значения соответствующей переменной равным Nothing (листинг 4.3, *б*) или когда программа выходит из области видимости переменной. В данном коде при обработке события Terminate генерируется подтверждающее сообщение со звуковым сигналом.

Листинг 4.3, *а*. Модуль класса Client

```
Public Name As String
Public Account As Long

Private Sub Class_Initialize()
```

```
   Name = "Ivanov"
   Account = 0
End Sub

Private Sub Class_Terminate()
   Beep
   MsgBox "Object killed"
End Sub
```

Листинг 4.3, *б*. Создание экземпляра класса. Стандартный модуль

```
Sub TestClient2()
   Dim c As New Client
   Debug.Print c.Name & vbTab & c.Account
   Set c = Nothing
End Sub
```

Методы и свойства

Методы класса обычно содержат код, который анализирует состояние объекта и изменяет его. Например, классы часто обладают определенными функциями, которые не сводятся к простому чтению или установке значений некоторых параметров, а требуют проведения определенных вычислений. В этом случае на помощь и приходят методы. По своей сути методы представляют собой процедуры. В том случае, если они возвращают или устанавливают значения, их удобнее программировать в виде функций. Вызов метода представляет собой операцию, выполняемую с объектом посредством ссылки на него, затем указания точки, после которой идет имя метода, за которым в скобках перечисляется список фактических параметров.

```
object.method(arglist)
```

Многие классы имеют открытые поля (public), к которым программисты могут обращаться напрямую, но в большинстве случаев такой подход оказывается не слишком удобным. Более подходящим является использование закрытых полей (private), т. е. таких полей, которые недоступны вне данного класса. Использование закрытых полей позволяет защищать данные от внешнего воздействия. Поэтому процесс получения и установления значений параметров текущего состояния объекта желательно реализовать через свойства или специальные методы. Для этого при описании свойства надо воспользоваться специальными процедурами `Property`:

- процедура `Property Let` служит для объявления имен свойств, значениями которых являются числовые данные;

- процедура `Property Set` служит для объявления имен свойств, значениями которых являются объекты;
- процедура `Property Get` обеспечивает возможность считывания значения свойств.

В следующем коде (листинг 4.4, *а*) расширяются функциональные возможности класса `Client` путем добавления в него четырех методов и двух свойств. В этом классе имеются два скрытых поля `mName` и `mAccount`, которые описывают имя и деньги на счете. Методы `Deposite` и `Withdraw` моделируют вложение и снятие денег со счета. Методы `ToString` и `ToDebug` позволяют представить информацию о клиенте в виде строки и отобразить ее в окне **Immediate**. Свойства `Name` и `Account` осуществляют доступ к закрытым полям `mName` и `mAccount`. Листинг 4.4, *б* демонстрирует создание экземпляра класса `Client` и использование его методов.

Листинг 4.4, *а*. Свойства. Модуль класса Client

```
Private mName As String
Private mAccount As Long

Property Get Name() As String
   Name = mName
End Property

Property Let Name(ByRef newName As String)
   mName = newName
End Property

Property Get Account() As String
   Account = mAccount
End Property

Property Let Account(ByRef newAccount As String)
   mAccount = newAccount
End Property

Public Sub Deposite(value As Long)
   mAccount = mAccount + value
End Sub

Public Sub Withdraw(value As Long)
```

```
   If value <= mAccount Then
      mAccount = mAccount - value
   End If
End Sub

Public Function ToString() As String
   ToString = "Name: " & mName & ", Account: " & mAccount
End Function

Public Sub ToDebug()
   Debug.Print Me.ToString
End Sub
```

Листинг 4.4, *б*. Свойства. Экземпляр класса Client

```
Sub TestClient()
   Dim c As New Client
   c.Name = "Bond"
   c.Account = 300000
   Debug.Print c.ToString
   c.Deposite 1000
   c.ToDebug
   c.Withdraw 2000
   c.ToDebug
End Sub
```

VBA позволяет создавать как свойства "только для чтения", т. е. те, которые могут только считывать данные, так и свойства "только для записи", т. е. те, которые могут только устанавливать значения. При описании свойства "только для чтения" не нужно объявлять процедуру блок `Property Let` или `Property Set`, а при описании свойства "только для записи" — блок `Property Get`.

События

До сих пор объекты, которые мы создавали, были глухими и немыми, т. е. они как не получали сообщений от других объектов, так и не передавали их, что может навести на мысль, что в VBA программы представляют собой последовательность выполняемых друг за другом процедур. Это совсем не так. Зачастую момент выполнения процедуры в программе обусловлен внешним

фактором — *событием* (*event*). Например, нажатием кнопки, выбором переключателя, сворачиванием формы и т. д. Таким образом, событие играет в приложении роль сигнала, информирующего о том, что в системе произошло что-то важное, требующее дополнительного внимания.

Событие объявляется в пределах класса при помощи ключевого слова `Event`. Объявление должно включать идентификатор события и список его параметров, например, у следующего события `MoneyWithdrawn`, которое может генерироваться при снятии денег со счета, имеются два параметра: `Money` и `Cancel`. Первый из них возвращает снимаемую сумму, а второй — определяет, надо ли отменить проводимую денежную операцию.

```
Public Event MoneyWithdrawn(ByVal Money As Long, ByRef Cancel As Boolean)
```

Сами по себе события не могут возвращать данные. Данные возвращаются через параметры события.

Объявление события означает только то, что оно может быть сгенерировано, в то время как сам процесс генерации события производится оператором `RaiseEvent`. Например, следующий оператор, помещенный в код, говорит о том, что при выполнении программы по его достижению сгенерируется событие `MoneyWithdrawn`.

```
RaiseEvent MoneyWithdrawn(Money, Cancel)
```

При объявлении переменной, которой будет присвоено значение экземпляра класса, имеющего право генерировать событие, надо дополнительно добавить ключевое слово `WithEvents`. Например, в приводимом ниже коде событие будет генерироваться у переменной `c` типа `Client`.

```
Private WithEvents c As Client
```

Для того чтобы отреагировать на событие, его надо ассоциировать со специальной процедурой, которая называется *обработчиком события* (event handlers). Такая ассоциация производится, используя шаблон процедуры обработки события.

```
Sub object_event(arglist)
...
End Sub
```

В рассматриваемом примере это будет следующая процедура.

```
Private Sub c_BigBucksDeposite(ByVal s As Double)
```

Приведем простой пример (листинги 4.5, *а, б*, *с*). В классе `Client` описаны два события:

- событие `BigBucksDeposite`, генерируемое при взносе на счет не меньше 10000 денежных единиц. Параметр этого события возвращает вносимую сумму;
- событие `GetOff` генерируется при попытке снять со счета сумму, превосходящую имеющуюся в наличии. Параметром события является экземпляр

класса `MoneyInfo`, который инкапсулирует в себе данные о размере счета и снимаемых деньгах.

Листинг 4.5, *а*. Класс с событием. Модуль класса `Client`

```
Public Event BigBucksDeposite(ByVal s As Long)
Public Event GetOff(ByVal e As MoneyInfo)

Private mName As String
Private mAccount As Long

Property Get Name() As String
   Name = mName
End Property

Property Let Name(ByRef newName As String)
   mName = newName
End Property

Property Get Account() As String
   Account = mAccount
End Property

Property Let Account(ByRef newAccount As String)
   mAccount = newAccount
End Property

Public Sub Deposite(value As Long)
   If value >= 10000 Then RaiseEvent BigBucksDeposite(value)
   mAccount = mAccount + value
End Sub

Public Sub Withdraw(value As Long)
   If value <= mAccount Then
      mAccount = mAccount - value
   Else
      Dim e As New MoneyInfo
      e.Account = mAccount
      e.Withdraw = value
```

```
      RaiseEvent GetOff(e)
   End If
End Sub
```

Листинг 4.5, *б*. Класс с событием. Модуль класса MoneyInfo

```
Public Account As Long
Public Withdraw As Long
```

Листинг 4.5, *с*. Класс с событием. Модуль листа или модуль ЭтаКнига

```
Private WithEvents c As Client

Sub TestClient()
   Set c = New Client
   c.Name = "Bond"
   c.Account = 300000
   c.Deposite 100000
   c.Withdraw 2000000
End Sub

Private Sub c_BigBucksDeposite(ByVal s As Long)
   Debug.Print "Deposite: " & s
End Sub

Private Sub c_GetOff(ByVal e As MoneyInfo)
   Debug.Print "Accounte: " & e.Account & ", Withdraw: " & e.Withdraw
End Sub
```

Таймер, как пример класса, генерирующего события

В качестве еще одного примера создадим класс `TimerState`, моделирующий таймер. У этого класса имеются два открытых поля — `Duration` и `Enabled`, задающие продолжительность работы таймера и его возможность выполнять задание. Кроме того, в классе имеется метод `TimerTask`, который запускает таймер. По завершении каждой 1/100 доли секунды в таймере генерируется событие `Tick`, а по завершении задания — событие `Collapse`.

Для испытания этого класса создайте форму, в которой расположите два поля ввода и две кнопки (**Start** и **Stop**), а в модуле формы наберите приводимый ниже код (листинги 4.6, *а*, *б*). Нажатие кнопки **Start** обеспечит включение таймера на 5 с. В первое поле ввода будет выводиться число прошедших с момента включения 1/100 долей секунды. Во второе поле ввода — сообщение о том, что таймер еще работает, либо прекратил работу, если такое событие случилось. Нажатие кнопки **Stop** вызовет остановку работы таймера.

Листинг 4.6, *а*. Таймер. Модуль класса `TimerState`

```
Public Duration As Double
Public Enabled As Boolean
Public Event Collapse()
Public Event Tick()

Public Sub TimerTask()
   Dim startTime As Double
   Dim eachSec As Double
   eachSec = 0
   startTime = Timer
   Do While Timer < startTime + Duration
      If Not Enabled Then Exit Sub
      If Timer - eachSec >= 1 Then
         eachSec = eachSec + 1
         DoEvents
         RaiseEvent Tick
      End If
   Loop
   RaiseEvent Collapse
End Sub
```

Листинг 4.6, *б*. Таймер. Модуль формы

```
Private WithEvents ts As TimerState
Private i As Integer

Private Sub UserForm_Initialize()
   Me.Caption = "Timer"
   CommandButton1.Caption = "Start"
```

```
    CommandButton2.Caption = "Stop"
    Set ts = New TimerState
End Sub

Private Sub CommandButton1_Click()
    TextBox2.Text = "Count"
    ts.Duration = 5
    ts.Enabled = True
    ts.TimerTask
End Sub

Private Sub CommandButton2_Click()
    ts.Enabled = False
    TextBox2.Text = "End"
    i = 0
End Sub

Private Sub ts_Tick()
    TextBox1.Text = i
    DoEvents
    i = i + 1
End Sub

Private Sub ts_Collapse()
    TextBox1.Text = ""
    TextBox2.Text = "End"
    i = 0
    DoEvents
End Sub
```

Объект *Collection*

Объект `Collection` позволяет динамически группировать семейство объектов, идентифицируя их по индексу. Объект `Collection` имеет лишь одно свойство `Count`, возвращающее число элементов набора, и три метода, приведенные в табл. 4.1.

Таблица 4.1. *Методы объекта* `Collection`

Метод	Описание
`Add`	Добавляет новый элемент в семейство. `Add item, key, before, after` *`item`* — обязательный параметр, специфицирующий добавляемый элемент; *`key`* — необязательный параметр, задающий параметр, идентифицирующий элемент. Может использоваться вместо его порядкового номера; *`before`* — необязательный параметр, указывающий элемент, перед которым размещается добавляемый элемент; *`after`* — необязательный параметр, определяющий элемент, после которого размещается добавляемый элемент
`Item`	Возвращает специфицированный элемент семейства. `Item(`*`index`*`)` Здесь *`index`* — порядковый номер элемента в семействе или его идентификатор, заданный параметром *`key`* метода `Add`. Нумерация элементов в объекте `Collection` начинается с единицы
`Remove`	Удаляет элемент из семейства. `Remove` *`index`* Здесь *`index`* — порядковый номер элемента в семействе или его идентификатор, заданный параметром *`key`* метода `Add`

В следующем коде (листинг 4.7, *а*) создается коллекция объектов `Client`, затем информация о ее элементах выводится в окно отладки. После чего из коллекции удаляется второй элемент, в окно отладки выводится уже видоизмененная коллекция, а затем при помощи функции `ItemExists` проверяется существование в коллекции указанных элементов. Для реализации этой функции в классе `Client` дополнительно определяется метод `Equal`, который сравнивает данный объект с указанным (листинг 4.7, *б*).

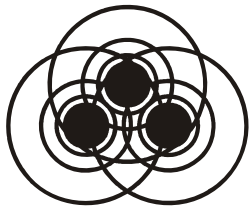
Листинг 4.7, *а*. Пример коллекции. Код из стандартного модуля

```
Sub DemoCollection()
   Dim li As New Collection
   Dim c1 As Client
   Set c1 = New Client
   c1.Account = 10000
   c1.Name = "Bond"
   li.Add c1
```

```
   Dim c2 As Client
   Set c2 = New Client
   c2.Account = 20000
   c2.Name = "Pooh"
   li.Add c2
   Dim i As Integer
   For i = 1 To li.Count
      Debug.Print li.Item(i).Name & " " & li.Item(i).Account
   Next
   li.Remove 2
   For i = 1 To li.Count
      Debug.Print li.Item(i).Name & " " & li.Item(i).Account
   Next
   Debug.Print ItemExists(li, c1)
   Debug.Print ItemExists(li, c2)
End Sub

Private Function ItemExists(NameOfCollection As Collection, _
                            LookupItem As Client)
   Dim Result As Boolean, i As Integer
   ItemExists = False
   For i = 1 To NameOfCollection.Count
      If LookupItem.Equal(NameOfCollection(i)) Then
            ItemExists = True
            Exit Function
      End If
   Next
End Function
```

Листинг 4.7, *б*. Пример коллекции. Код из модуля класса `Client`

```
Public Name As String
Public Account As Long

Public Function Equal(val As Client)
   If Me.Name = val.Name And Me.Account = val.Account Then
      Equal = True
   Else
      Equal = False
   End If
End Function
```



Глава 5

Работа со строками, временем и датами

Функции обработки строк

В VBA имеются следующие функции обработки строковых выражений, которые позволяют произвести широкий спектр операций: от нахождения длины строки, до замены в строке подстрок:

`Asc`	`Chr`	`LCase`	`UCase`
`Left`	`Right`	`Mid`	`Len`
`LTrim`	`RTrim`	`Trim`	`Space`
`Split`	`Join`	`String`	`StrComp`
`StrReverse`	`InStr`	`InStrRev`	`Replace`
`StrConv`			

Все эти функции с примерами будут подробно описаны в следующих подразделах.

Нахождение ASCII-кода по литералу и литерала по ASCII-коду

Функция `Asc` возвращает ASCII-код начальной буквы строки. Например, две следующие инструкции выведут 72 в окно **Immediate**.

```
Debug.Print Asc("Hello")
Debug.Print Asc("H")
```

Функция `Chr` преобразует ASCII-код в строку.

```
Chr(Charcode)
```

Здесь *`Charcode`* принимает значения от 0 до 255. Значения от 0 до 31 соответствуют так называемым управляющим кодам. Например, `Chr(13)` возвращает символ перехода на следующую строку, а `Chr(97)` возвращает символ "`a`".

Для обозначения некоторых наиболее чисто употребляемых клавиш существуют встроенные константы VBA, например, для клавиши <Backspace> — vbBack, клавиши <Tab> — vbTab, клавиши <Enter> — vbCr.

В следующем примере в окне в первой строчке будет отображаться Visual Basic, а во второй — for Applications:

```
MsgBox "Visual Basic" & vbCr & "for Applications"
```

Преобразование строки к нижнему или верхнему регистру

Функции LCase и UCase преобразуют строку к нижнему и верхнему регистрам. Например, следующие инструкции отобразят HELLO и hello.

```
Debug.Print LCase("Hello")
Debug.Print UCase("Hello")
```

Как сделать так, чтобы каждое слово в предложении начиналось с заглавной буквы

Функция StrConv может произвести более тонкие преобразования строки, чем функции LCase и UCase. Если значение ее второго параметра равно vbUpperCase или vbLowerCase, то все буквы переводятся в верхний или нижний регистр, а если равно vbProperCase, то в верхний регистр переводятся первые буквы каждого слова строки.

```
Dim strName As String
strName = "андрей гарнаев"
strName = StrConv(strName, vbProperCase)
MsgBox strName ' Отобразится Андрей Гарнаев
```

Возвращение подстроки по указанному местоположению

Функции Left и Right возвращают подстроки, состоящие из заданного числа соответственно первых и последних символов данной строки. Функция Mid возвращает подстроку строки, содержащую указанное число символов, начиная со специфицированной позиции. Например:

```
Debug.Print Left("Hello, World", 5) ' Отобразится Hello
Debug.Print Right("Hello, World", 5) ' Отобразится World
Debug.Print Mid("Hello, World", 8, 2)
' Отобразится Wo, т. к. два символа, начиная с 8-й позиции
```

Определение длины строки

Функция `Len` определяет длину строки. Например, следующий код выводит данную строку посимвольно.

```
Dim i As Integer
Dim s As String
s = "Hello, World"
For i = 1 To Len(s)
   Debug.Print Mid(s, i, 1)
Next
```

Копия строки без начальных и конечных пробелов

Функции `LTrim`, `RTrim` возвращают копию строки без пробелов соответственно в ее начале и конце, а функция `Trim` — без пробелов, как в начале, так и в конце строки. Например, в следующем коде переменной `sl` будет присвоено значение `"Hello, World  "`, переменной `sr` — значение `"  Hello, World"`, а переменной `st` — значение `"Hello, World"`.

```
Dim s As String, sl As String, sr As String, st As String
s = "   Hello, World   "
sl = LTrim(s)
sr = RTrim(s)
st = RTrim(s)
```

Строка, состоящая из указанного числа пробелов

Функция `Space` возвращает строку, состоящую из указанного числа пробелов. Например, в следующем коде (листинг 5.1) за счет последовательного добавления перед данной строкой другой, состоящей каждый раз из меньшего числа пробелов, создается эффект бегущей строки.

Листинг 5.1. Бегущая строка

```
Private n As Integer

Sub LetsGo()
   n = 10
   RunString
End Sub

Sub RunString()
```

```
   If n >= 0 Then
      Range("A1").Value = Space(n) & "Hello, World!"
      n = n - 1
      Application.OnTime Now + TimeValue("00:00:01"), "RunString"
   End If
End Sub
```

Строка, состоящая из указанного числа повторяющихся символов

Функция `String` возвращает строку, состоящую из указанного числа повторений одного и того же символа.

`String(`*`Number`*`, `*`Character`*`)`, где:

- *`Number`* — число повторений символа;
- *`Character`* — повторяемый символ.

Например, `String(10, "O") & "!"` возвращает значение "`OOOOOOOOOO!`".

Преобразование строки в массив

Функция `Split` преобразует строку в массив.

`Split(`*`Expression`*`, [`*`Delimiter`*`], [`*`Limit`*`], [`*`Compare`*`])`, где:

- *`Expression`* — преобразуемое строковое выражение;
- *`Delimiter`* — разделитель между элементами строкового выражения, из которых создается массив. По умолчанию используется пробел;
- *`Limit`* — максимальное число элементов в массиве. Если значение параметра равно −1, то нет ограничения на размерность массива;
- *`Compare`* — целое число, указывающее критерий отбора строк.

Например, следующий код разобьет строку на составляющие ее слова и предлоги.

```
Dim s As String
Dim m As Variant
Dim i As Integer
s = "Алиса в стране чудес"
m = Split(s)
For i = LBound(m) To UBound(m)
   Debug.Print m(i)
Next
```

Преобразование массива в строку

Функция `Join` преобразует массив в строку. При этом в качестве значения необязательного параметра можно указать разделитель между элементами строкового выражения, из которых создается массив. По умолчанию используется пробел. Например:

```
Dim m As Variant
Dim s As String
m = Array("Вера", "Надежда", "Любовь")
s = Join(m, ", ")
```

В результате переменной `s` будет присвоено значение "`Вера, Надежда, Любовь`".

Сравнение строк

Функция `StrComp` возвращает результат сравнения двух строк.

`StrComp(`*`String1`*`, `*`String2`*` [, `*`Compare`*`])`, где:

- *`String1`* и *`String2`* — два любых строковых выражения;
- *`Compare`* указывает способ сравнения строк. Допустимые значения: `vbBinaryCompare` или 0 (двоичное сравнение), `vbTextCompare` или 1 (посимвольное сравнение без учета регистра).

Функция `StrComp` возвращает следующие значения:

- если *`String1`* меньше чем *`String2`*, то −1;
- если *`String1`* равняется *`String2`*, то 0;
- если *`String1`* больше чем *`String2`*, то 1.

Например:

```
Debug.Print StrComp("Hello", "hello", vbBinaryCompare)
' Возвращает -1
Debug.Print StrComp("Hello", "Hello", vbTextCompare)
' Возвращает 0
Debug.Print StrComp("Hello", "He", vbBinaryCompare)
' Возвращает 1
Debug.Print StrComp("Hello", "Hello, World", vbTextCompare)
' Возвращает -1
```

Создание зеркальной строки по отношению к данной

Функция `StrReverse` возвращает строку, состоящую из тех же символов, что и данная, но расположенных в обратном порядке. Например:

```
Debug.Print StrReverse("Hello") ' Возвращает olleH
```

Нахождение вхождения в строку подстроки

Функции `InStr`, `InStrRev` возвращают позиции первого вхождения подстроки в данную строку, начиная соответственно с ее начала и с конца. Если вхождения нет, то возвращается значение 0.

InStr([Start,] *StringCheck*, *StringMatch*[, *Compare*])

InStrRev(*StringCheck*, *StringMatch*, [*Start*], [*Compare*]), где:

- *Start* — числовое выражение, задающее позицию, с которой начинается каждый поиск. Если этот параметр опущен, то поиск начинается с первого символа строки;
- *StringCheck* — строковое выражение, в котором выполняется поиск;
- *StringMatch* — искомое строковое выражение;
- *Compare* — указывает способ сравнения строк. Допустимые значения: `vbBinaryCompare` или 0 (для двоичного сравнения), `vbTextCompare` или 1 (посимвольное сравнение без учета регистра).

```
Debug.Print InStr("Hello", "lo") ' Возвращает 4
Debug.Print InStr("Hello", "wo") ' Возвращает 0
```

Замена в строке подстроки

Функция `Replace` заменяет в строке подстроку.

Replace(*Expression*, *Find*, *Replacewith*[, *Start*[, *Count* [, *Compare*]]]), где:

- *Expression* — строка, в которой заменяется подстрока;
- *Find* — заменяемая подстрока;
- *Replacewith* — подстрока, на которую заменяется подстрока, указанная в качестве значения параметра *Find*;
- *Start* — позиция в строке, с которой ищется подстрока, указанная в качестве значения параметра *Find*. Если этот параметр опущен, то поиск производится с первой позиции;
- *Count* — указывает, сколько найденных подстрок надо заменить. Если параметр опущен, то будут произведены все замены;
- *Compare* — критерий сравнения при поиске подстроки. Допустимые значения: 0 (для двоичного сравнения), 1 (посимвольное сравнение без учета регистра).

```
Dim s As String, r As String
s = "Как хороши, как свежи были розы"
r = Replace(s, "как", "так", , , 1)
```

В результате получится строка "так хороши, так свежи были розы".

Замена всех специфицированных подстрок в рабочем листе

Метод `Replace` объекта `Range` позволяет произвести замену подстроки в указанной ячейке. Если организовать перебор всех ячеек, то это как раз и реализует замену подстроки во всех ячейках рабочего листа.

```
Sub ReplaceInfo(oldString As String, newString As String)
   Dim sh As Worksheet
   For Each sh In Worksheets
      sh.Cells.Replace What:=oldString, _
                       Replacement:=newString, _
                       LookAt:=xlPart, _
                       MatchCase:=False
   Next
End Sub
```

Функции времени и даты

В VBA имеются следующие функции времени и даты, которые позволяют произвести широкий спектр операций от определения текущей даты до сложения указанных дат:

`Date`	`DateAdd`	`DateDiff`	`DatePart`
`DateSerial`	`Day`	`Hour`	`Minute`
`Month`	`Now`	`Second`	`Time`
`Timer`	`TimeSerial`	`TimeValue`	`Weekday`
`Year`			

В последующих разделах на примерах рассмотрены все эти функции.

Определение текущей системной даты

Функция `Date` возвращает значение типа `Variant` (`Date`), содержащее текущую системную дату. Например, данная строка писалась 12 августа 2003 г., поэтому инструкция:

```
Debug.Print Date
```

выводит значение `12.08.2003`.

Определение текущего системного времени

Функция `Time` возвращает значение типа `Variant` (`Date`), содержащее текущее время по системным часам компьютера. Например, данная строка

писалась в 20 часов 46 минут и 14 секунд 12 августа 2002 г., поэтому инструкция:

```
Debug.Print Time
```

выводит значение `20:46:14`.

Определение текущей даты и системного времени

Функция `Now` возвращает значение типа `Variant` (`Date`), содержащее текущую дату и время по системному календарю и часам компьютера. Например, данная строка писалась в 20 часов 49 минут и 24 секунды 12 августа 2002 г., поэтому инструкция:

```
Debug.Print Now
```

выводит значение `12.08.2002 20:49:24`.

Извлечение из времени часового, минутного и секундного компонентов

Функции `Hour`, `Minute`, `Second` возвращают значения типа `Variant` (`Integer`), являющиеся целыми числами и представляющие часы, минуты и секунды в значении времени. В следующем примере переменным `h`, `m` и `s` присваиваются значения `16`, `35` и `17`.

```
Dim dt As Date
Dim h, m, s As Integer
dt = #4:35:17 PM#
h = Hour(dt)
m = Minute(dt)
s = Second(dt)
```

Извлечение из даты годового, месячного и дневного компонентов

Функции `Day`, `Month`, `Year` возвращают значения типа `Variant` (`Integer`), содержащие целые числа, которые представляют день, месяц, год в значении даты.

В данном примере, переменным `d`, `m` и `y` присваиваются значения `17`, `5` и `1960`.

```
Dim dt As Date
Dim d, m, y As Integer
dt = #5/17/1960#
d = Day(dt)
m = Month(dt)
y = Year(dt)
```

Определение дня недели

Функция `Weekday` возвращает значение типа `Variant` (`Integer`), содержащее целое число, представляющее день недели.

`Weekday(`*`Date`*`, [`*`Firstdayofweek`*`])`, где:

- *`Date`* — выражение, представляющее дату;
- *`Firstdayofweek`* — указывает первый день недели. Если этот параметр опущен, то подразумевается значение `vbSunday` (воскресенье). Допустимы также значения: `vbMonday` (понедельник), `vbTuesday` (вторник), `vbWednesday` (среда), `vbThursday` (четверг), `vbFriday` (пятница) и `vbSaturday` (суббота).

В данном примере, переменной `dw` присваивается `3`, т. е. 17 мая 1960 г. было вторником.

```
Dim dt As Date
Dim dw As Integer
dt = #5/17/1960#
dw = Weekday(dt, vbMonday)
```

Определение числа секунд, прошедших с полуночи

Функция `Timer` возвращает значение типа `Single`, представляющее число секунд, прошедших после полуночи. В Windows, в отличие от Macintosh, функция `Timer` возвращает даже не число секунд, а число долей секунд, прошедших после полуночи. Например, следующий код (листинг 5.2) создает мигающую ячейку, у которой цвет фона в течение 5 секунд попеременно становится то красным, то зеленым. Для того чтобы во время выполнения цикла компьютер не зависал, и приложение могло реагировать на различные события, происходящие в системе, в циклы добавлены операторы `DoEvents`.

Листинг 5.2. Мигающая ячейка

```
Sub LetsGo()
   Dim fl As Boolean
   Dim old As Long
   old = Timer
   Do
      fl = Not fl
      If fl Then
         Range("A1").Interior.Color = RGB(255, 0, 0)
      Else
         Range("A1").Interior.Color = RGB(0, 255, 0)
```

```
      End If
      Delay
      DoEvents
   Loop While Timer - old <= 5
   Range("A1").Interior.ColorIndex = xlNone
End Sub

Sub Delay()
   Dim old As Long
   old = Timer
   Do
     DoEvents
   Loop While Timer - old <= 0.5
End Sub
```

Определение количества лет, кварталов, месяцев, недель и дней, прошедших между двумя датами

Функция `DateDiff` возвращает значение типа `Variant` (`Long`), показывающее временной интервал между двумя датами.

`DateDiff(`*`Interval`*`, `*`Date1`*`, `*`Date2`*`[, `*`Firstdayofweek`*`[, `*`Firstweekofyear`*`]])`, где:

- *`Interval`* — строковое выражение, указывающее тип временного интервала, который следует использовать при вычислении разности между датами *`Date1`* и *`Date2`*. Допустимые значения: `yyyy` (год), `q` (квартал), `m` (месяц), `y` (день года), `d` (день месяца), `w` (день недели), `ww` (неделя), `h` (часы), `m` (минуты), `s` (секунды);
- *`Date1`*, *`Date2`* — значения типа `Variant (Date)`. Две даты, разность между которыми следует вычислить;
- *`Firstdayofweek`* — константа, указывающая первый день недели;
- *`Firstweekofyear`* —константа, указывающая первую неделю года.

В данном примере переменной `dayspast` присваивается `507`, т. е. между числом, когда писалась эта строка (12 августа 2002 г.) и 17 мая 1960 г. прошло 507 месяцев.

```
Dim dayspast As Long
dayspast = DateDiff("m", #5/17/1960#, Now)
```

В качестве еще одного примера приведем функцию, вычисляющую число целых лет, прошедших со дня рождения. Например, сегодня 6 декабря 2003 г. Тогда функция `WholeYears(#5/17/1960#)` возвращает значение 43.

```
Function WholeYears(birthDate As Date) As Long
   Dim age As Long
```

```
    If Not IsDate(birthDate) Then
       birthDate = Date
    End If
    If birthDate > Date Then
       birthDate = Date
    End If
    age = DateDiff("yyyy", birthDate, Date)
    If DateSerial(Year(birthDate), Month(birthDate), _
                  Day(birthDate)) > Date Then
       age = age - 1
    End If
    WholeYears = age
End Function
```

Определение компонента даты

Функция `DatePart` возвращает значение типа `Variant` (`Integer`), содержащее указанный компонент даты.

`DatePart(`*`Interval`*`, `*`Date`*`[, `*`Firstdayofweek`*`[, `*`Firstweekofyear`*`]])`, где:

- *`Interval`* — строковое выражение, указывающее тип временного интервала, который должен быть найден. Допустимые значения: `yyyy` (год), `q` (квартал), `m` (месяц), `y` (день года), `d` (день месяца), `w` (день недели), `ww` (неделя), `h` (часы), `m` (минуты), `s` (секунды);
- *`Date`* — значения типа `Variant (Date)`, специфицированную компоненту которого возвращает данная функция;
- *`Firstdayofweek`* — константа, указывающая первый день недели;
- *`Firstweekofyea`* — константа, указывающая первую неделю года.

В данном примере переменной pt присваивается `2`, т. е. 17 мая 1960 г. было во втором квартале 1960 г.

```
Dim pt As Integer
pt = DatePart("q", #5/17/1960#, Now)
```

Добавление к дате указанного временного интервала

Функция `DateAdd` возвращает значение типа `Variant` (`Date`), содержащее дату, к которой добавлен указанный временной интервал.

`DateAdd(`*`Interval`*`, `*`Number`*`, `*`Date`*`)`, где:

- *Interval* — строковое выражение, указывающее тип добавляемого временного интервала. Допустимые значения: yyyy (год), q (квартал), m (месяц), y (день года), d (день месяца), w (день недели), ww (неделя), h (часы), m (минуты), s (секунды);
- *Number* — числовое выражение, указывающее число временных интервалов, которое следует добавить. Оно может быть положительным (для получения более поздних дат) или отрицательным (для получения более ранних дат);
- *Date* — значение типа Variant (Date) или литерал даты, представляющий дату, к которой добавляется указанный временной интервал.

В данном примере к дате 05/17/60 прибавляется 100 месяцев. В результате переменной d будет присвоено значение 17.09.1968.

```
Dim d As Date
d = DateAdd("m", 100, #05/17/60#)
```

Преобразование часов, минут и секунд в формат времени

Функция TimeSerial возвращает значение типа Variant (Date), содержащее значение времени, соответствующее указанному часу, минуте и секунде.

```
TimeSerial(Hour, Minute, Second)
```

Здесь параметры *Hour*, *Minute* и *Second* принимают значения типа Integer. В данном примере переменной t присваивается значение 16:35:17.

```
Dim t As Date
t = TimeSerial(16, 35, 17)
```

Преобразование года, месяца и дня в формат даты

Функция DateSerial возвращает значение типа Variant (Date), соответствующее указанному году, месяцу и дню.

```
DateSerial(Year, Month, Day)
```

Здесь параметры *Year*, *Month* и *Day* принимают значения типа Integer. В данном примере переменной d присваивается значение 17.05.1960.

```
Dim d As Date
d = DateSerial(1960, 5, 17)
```

Преобразование строки в формат времени

Функция TimeValue преобразует строку в формат времени. Например, в следующем коде строка "17:23:24" преобразуется в соответствующее время, из которого затем извлекается соответствующее число часов.

```
Dim d As Date
d = TimeValue("17:23:24")
Debug.Print Hour(d)
```

Примечание

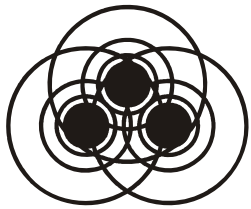
Время, конечно можно было бы также задать, используя литералы времени, а именно `d = #17:23:24#`.

Проверка, является ли год високосным

В VBA элементарно просто проверить, является ли год високосным. Для этого надо воспользоваться функцией `IsLeapYear`, которая проверяет, является ли указанное значение датой, а затем ее проверить на 29 февраля тестируемого года.

```
Public Function IsLeapYear(intYear As Integer) As Boolean
   IsLeapYear = IsDate("2/29/" & intYear)
End Function
```

Глава 6



Пользовательские функции

В данном разделе на нескольких примерах покажем, как строятся функции пользователя. Начнем с простейшего примера функции, вычисляющей значение по одной формуле, например, нахождению квадрата числа. Затем обсудим более сложные примеры, а именно функции, значения которых зависят от условий. Продемонстрируем, как пользовательские функции могут пригодиться при расчете комиссионных, определении числового формата ячейки, проверке вхождения данной строки в другую по образцу, определения Web-адреса по гиперссылке, содержащейся в ячейке. Рассмотрим, как создаются функции, параметрами которых являются диапазоны, функции с необязательными параметрами и неопределенным их количеством. Кроме того, будет объяснено, как конструируются надстройки.

Где пишется код функции пользователя?

Сначала спросим себя, где строятся пользовательские функции. Ответ: в стандартном модуле редактора VBA. Для того чтобы попасть в редактор VBA выберите команду **Сервис** | **Макрос** | **Редактор Visual Basic** или нажмите комбинацию клавиш <Alt>+<F11>. В результате вы попадете в интегрируемую среду разработки приложений IDE редактора Visual Basic. Она имеет стандартный вид для Windows-приложений: строка меню, панель инструментов (в данном случае **Standard**) и два окна **Project — VBA Project** и **Properties**. Пока нас интересует только окно **Project — VBA Project**. В нем отображается реестр модулей и форм, входящих в создаваемый проект. Значок активного модуля в окне **Project — VBA Project** выделяется серым цветом. Единственный модуль, который нас интересует в данной главе, это стандартный модуль. Для того чтобы его добавить в проект, выберите команду **Insert** | **Module**.

Ваша первая функция пользователя

Итак, теперь можно перейти к первому примеру построения функции пользователя. Давайте, для начала построим очень простую функцию, которая

возвращает квадрат введенного значения. В стандартном модуле наберите код из листинга 6.1 (рис. 6.1).

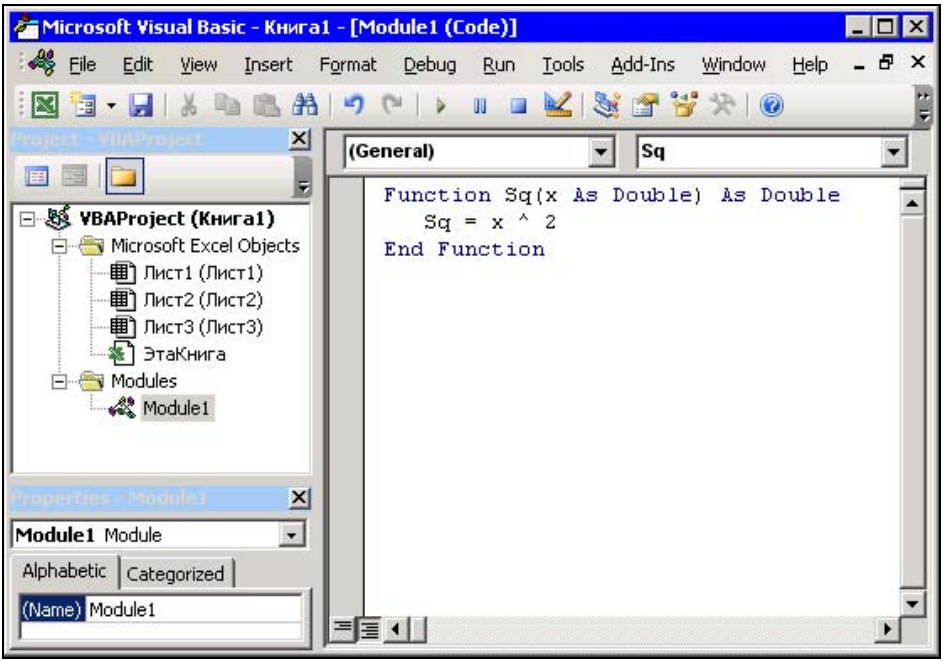


Рис. 6.1. Код пользовательской функции в модуле

Листинг 6.1. Пользовательская функция

```
Function Sq(x As Double) As Double
   Sq = x ^ 2
End Function
```

Вот, собственно говоря, и все, пользовательская функция создана. По умолчанию она попадает в раздел **Определенные пользователем** списка **Категория** окна **Мастер функций**. Найдем, например, квадрат значения 2.5. Для этого:

1. Выберите ячейку `A2`.
2. Введите число 2,5 в ячейку `A2`.
3. Выберите ячейку `B2`, в которой найдем значение функции.
4. Выберите команду **Вставка** | **Функция**.
5. В первом окне Мастера функций в списке **Категория** выберите **Определенные пользователем**, а в списке **Выберите функцию** укажите **Sq**. Нажмите кнопку **OK.**

6. Во втором окне Мастера функций (рис. 6.2) в поле **x** введите ссылку на ячейку `A2` и нажмите кнопку **OK**.

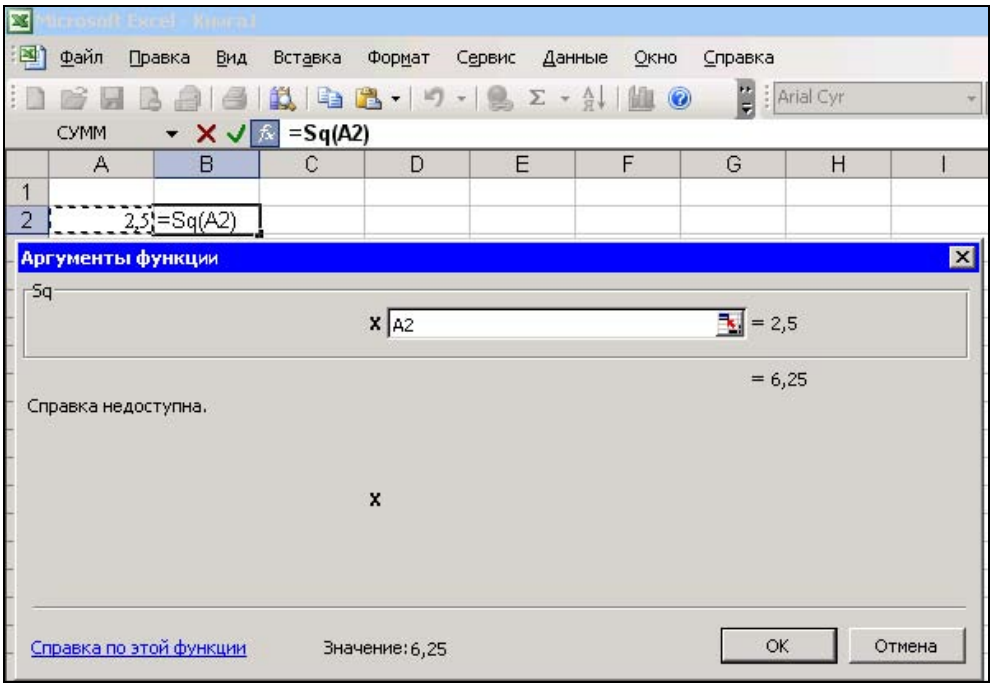


Рис. 6.2. Вычисление значения пользовательской функции при помощи Мастера функций

Математические функции

При создании функций пользователя полезно знать, какие встроенные математические функции имеются в VBA. На самом деле, таких функций совсем не много:

- `Abs(`*number*`)` — абсолютная величина числа;
- `Atn(`*number*`)` — арктангенс;
- `Cos(`*number*`)` — косинус;
- `Exp(`*number*`)` — экспонента, т. е. результат возведения основания натурального логарифма в указанную степень;
- `Log(`*number*`)` — натуральный логарифм;
- `Rnd(`*number*`)` — случайное число из интервала [0, 1);
- `Sgn(`*number*`)` — знак числа;

- `Sin(number)` — синус;
- `Sqr(number)` — квадратный корень из числа;
- `Tan(number)` — тангенс;
- `Fix(number)`, `Int(number)` — обе функции `Int` и `Fix` отбрасывают дробную часть числа и возвращают целое значение. Различие между функциями `Int` и `Fix` состоит в том, что для отрицательного значения параметра `число` функция `Int` возвращает ближайшее отрицательное целое число, меньшее либо равное указанному, а `Fix` — ближайшее отрицательное целое число, большее либо равное указанному.

Как найти значение числа π?

В VBA нет функции, возвращающей число π. Поэтому, для нахождения числа π применяется функция `Atn`, как показано в следующем примере (листинг 6.2, *а*):

Листинг 6.2, *а*. Число π

```
Function SinPi(x As Double) As Double
   SinPi = Sin(Pi() * x)
End Function

Function Pi() As Double
   Pi = Atn(1) * 4
End Function
```

Конечно, можно задать π и явным образом, указав достаточное число значащих цифр, но этот подход менее элегантен, чем с помощью функции `Atn` (листинг 6.2, *б*).

Листинг 6.2, *б*. Число π

```
Const Pi As Double = 3.14159265358979

Function SinPi(x As Double) As Double
   SinPi = Sin(Pi * x)
End Function
```

Определение числового формата ячейки

В качестве еще одного примера пользовательской функции с одним параметром приведем функцию (листинг 6.3), которая возвращает не результат вычислений, а числовой формат выбранной ячейки. Основную роль в этой функции выполняет свойство `NumberFormat` объекта `Range`, которое и возвращает искомое значение.

Листинг 6.3. Определение числового формата ячейки

```
Function GetNumberFormat(cell As Range) As String
   GetNumberFormat = cell.NumberFormat
End Function
```

Расчет комиссионных

Менеджерам по продажам часто приходится просчитывать комиссионные от продаж. В приводимой таблице комиссионные основаны на динамической шкале — чем больше продано, тем больше процент.

Объем продаж за неделю, руб.	Комиссионные, %
От 0 до 9 999	8
От 10 000 до 19 999	10
Более 20 000	14

Следующая пользовательская функция (листинг 6.4, *а*) решает задачу начисления комиссионных.

Листинг 6.4, *а*. Расчет комиссионных

```
Function Commission(Sales As Long) As Double
   Const Interest1 As Double = 0.08
   Const Interest2 As Double = 0.1
   Const Interest3 As Double = 0.14
   Const Sales1 As Long = 10000
   Const Sales2 As Long = 20000
   If Sales < Sales1 Then
      Commission = Sales * Interest1
   ElseIf Sales < Sales2 Then
      Commission = Sales * Interest2
```

```
   Else
      Commission = Sales * Interest3
   End If
End Function
```

Учитывая, что параметр функции `Commission` имеет тип `Long`, можно воспользоваться оператором выбора вместо оператора условного перехода и определить эту функцию следующим равносильным способом (листинг 6.4, *б*).

Листинг 6.4, *б*. Расчет комиссионных

```
Function Commission(Sales As Long) As Double
   Const Interest1 As Double = 0.08
   Const Interest2 As Double = 0.1
   Const Interest3 As Double = 0.14
   Const Sales1 As Long = 10000
   Const Sales2 As Long = 20000
   Select Case Sales
      Case Is < Sales1
         Commission = Sales * Interest1
      Case Is < Sales2
         Commission = Sales * Interest2
      Case Else
         Commission = Sales * Interest3
  End Select
End Function
```

Функция, имеющая несколько параметров

Пользовательская функция может иметь как один, так и несколько параметров. Например, приводимая в листинге 6.5 функция находит сумму двух чисел.

Листинг 6.5. Функция с двумя параметрами

```
Function SumXY(x As Double, y As Double) As Double
   SumXY = x + y
End Function
```

Функция, проверяющая вхождение данной строки в другую по образцу

В качестве еще одного примера функции с несколькими параметрами, создадим функцию (листинг 6.6), проверяющую вхождение данной строки в другую по образцу.

Листинг 6.6. Проверка вхождения данной строки в другую

```
Function IsLike(value As String, pattern As String) As Boolean
   If value Like pattern Then
      IsLike = True
   Else
      IsLike = False
   End If
End Function
```

Приведем примеры работы данной функции:

Значение в ячейке **A1**	Формула из ячейки **B1**	Возвращаемое значение
Excel	=IsLike(A1;"Exce?")	True
Exception	=IsLike(A1;"Exce?")	False
Excel	=IsLike(A1;"[aEiou]*")	True
Excel	=IsLike(A1;"[aEiou]*")	False

Функция, возвращающая Web-адрес из ячейки с гиперссылкой

Следующий пример (листинг 6.7) является функцией, возвращающей Web-адрес из ячейки, содержащий гиперссылку. Информация о гиперссылке инкапсулируется в объект `Hyperlink`, и в частности ее адрес возвращается свойством `Address`. В случае, если в ячейке содержится адрес электронной почты, из возвращаемой строки свойством `Address` надо удалить префикс `mailto:`

Листинг 6.7. Определение Web-адреса по гиперссылке

```
Function GetAddressFromHyperlinkCell(Cell As Range)
   GetAddressFromHyperlinkCell = _
      Replace(Cell.Hyperlinks(1).Address, "mailto:", "")
End Function
```

Расчет сложных комиссионных

Давайте немного усложним задачу расчета комиссионных. Будем считать, что комиссионные зависят от ставки, занимаемой менеджером. Если он принят в постоянный штат фирмы, то комиссионные начисляются по описанному выше закону. Если же он находится на испытательном сроке, то его комиссионные составляют 75% от номинала. Для решения этой задачи функции `Commission` надо добавить еще один аргумент `IsTemporal`, который и идентифицирует ставку менеджера (листинг 6.8). Если менеджер находится на испытательном сроке, то значение аргумента `IsTemporal` полагается равным `True`, а если он зачислен в постоянный штат, то значение этого аргумента полагается равным `False`. Тогда функция `Commission` примет следующий вид.

Листинг 6.8. Расчет комиссионных с учетом ставки менеджера

```
Function Commission(Sales As Long, IsTemporal As Boolean) As Double
   Const Interest1 As Double = 0.08
   Const Interest2 As Double = 0.1
   Const Interest3 As Double = 0.14
   Const Sales1 As Long = 10000
   Const Sales2 As Long = 20000
   Const TMPInterest As Double = 0.75
   Dim Payment As Double
   If Sales < Sales1 Then
      Payment = Sales * Interest1
   ElseIf Sales < Sales2 Then
      Payment = Sales * Interest2
   Else
      Payment = Sales * Interest3
   End If

   If IsTemporal Then
      Commission = TMPInterest * Payment
   Else
      Commission = Payment
   End If
End Function
```

Функция, параметрами которой является диапазон

Значениями параметров функций могут быть не только числа, но и ссылки на диапазоны. Например, сконструируем функцию, которая аналогично функции рабочего листа СУММ, находит сумму значений ячеек указанного диапазона (листинг 6.9). В конструируемой функции во время суммирования, когда считывается значение из очередной ячейки, происходит дополнительная проверка при помощи функции IsNumeric, содержит ли она число.

Листинг 6.9. Функция, параметрами которой является диапазон

```
Function Sum(rgn As Range) As Double
   Dim c As Range
   Dim s As Double
   For Each c In rgn
      If IsNumeric(c.Value) Then s = s + c.Value
   Next
   Sum = s
End Function
```

Подсчет числа ячеек со значениями из указанного диапазона

В качестве еще одного примера функции, параметром которой является ссылка на диапазон, сконструируем функцию, возвращающую число ячеек из данного диапазона со значениями, лежащими в указанных границах (листинг 6.10).

Листинг 6.10. Подсчет числа ячеек со значениями из указанного диапазона

```
Function CountValues(rgn As Range, LowBound As Double, _
                     UpperBound As Double) As Long
   Dim c As Range
   Dim k As Long
   k = 0
   For Each c In rgn
      If c.Value >= LowBound And c.Value <= UpperBound Then
         k = k + 1
```

```
      End If
   Next
   CountValues = k
End Function
```

Например, для того чтобы определить число ячеек из диапазона **B2:B100** со значениями от 1000 до 10000, в ячейку **C2** достаточно ввести следующую функцию:

```
=CountValues(B2:B100;1000;10000)
```

Функция с необязательными параметрами

Приведем пример функции с необязательными параметрами (листинг 6.11). Функция `UserName` возвращает строку, состоящую из имени и фамилии, указанных в качестве значений ее параметров. Если какой-то из параметров опущен, то она возвращает неполное имя. При работе с необязательными параметрами необходимо использовать функцию `IsMissing`, возвращающую значение `True`, если соответствующий параметр не был передан в процедуру, и `False`, в противном случае.

Листинг 6.11. Функция с необязательными параметрами

```
Function UserName(Optional LastName As String, _
                  Optional FirstName As String) As String
   If Not (IsMissing(LastName)) And Not (IsMissing(FirstName)) Then
      UserName = LastName & Space(1) & FirstName
   ElseIf IsMissing(LastName) And Not (IsMissing(FirstName)) Then
      UserName = FirstName
   ElseIf Not IsMissing(LastName) And IsMissing(FirstName) Then
      UserName = LastName
   End If
End Function

Sub DemoUserName()
   MsgBox UserName(LastName:="Bond", FirstName:="James")
   '  Bond James
   MsgBox UserName("Bond", "James")
   '  Bond James
   MsgBox UserName("Bond")
   '  Bond
   MsgBox UserName(, "James")
```

```
   '  James
   MsgBox UserName()
   ' Nothing
End Sub
```

Использование неопределенного числа параметров

Как правило, число передаваемых в процедуру параметров совпадает с числом определенных у этой процедуры параметров. Однако ключевое слово `ParramArray` предоставляет возможность ввода в процедуру произвольного, заранее не указанного числа параметров (например, как это происходит в функции `СУММ` рабочего листа в MS Excel). В качестве примера приведем процедуру `Function`, которая из строк, используемых в качестве значений параметров, образует одну строку (листинг 6.12).

Листинг 6.12. Составление предложения из слов

```
Function PutTogether(FirstWord As String, _
                     ParamArray Words() As Variant) As String
   Dim s As String
   Dim a As Variant
   s = FirstWord
   For Each a In Words
      s = s & a
   Next
   PutTogether = s
End Function

Sub DemoPutTogether()
   Dim a As String, b As String, c As String
   a = "Капля "
   b = "камень "
   c = "точит"
   MsgBox PutTogether("Пословица: ", a, b, ñ)
' Отобразится: Пословица: Капля камень точит
   MsgBox PutTogether("Поговорка: ", "Упорство ", _
                      "и труд", " все ", "перетрут")
'Отобразится: Поговорка: Упорство и труд все перетрут
End Sub
```

В качестве еще одного примера функции с неопределенным числом параметров создадим аналог функции рабочего листа СУММ, который позволяет суммировать как значения из выбранных диапазонов, так и введенные пользователем числа (листинг 6.13). В данном коде функция TypeName проверяет тип очередного значения параметра. Если таковым является диапазон, то суммирование производится по всем входящим в него ячейкам, а если число — то оно просто прибавляется к текущей сумме.

Листинг 6.13. Нахождение суммы элементов выделенного диапазона

```
Function ReturnSum(ParamArray rgns() As Variant) As Double
   Dim v As Double
   Dim r As Variant
   Dim c As Range
   For Each r In rgns
      If TypeName(r) = "Range" Then
         For Each c In r
            If IsNumeric(c.Value) Then v = v + c.Value
         Next
      ElseIf TypeName(r) = "Double" Or TypeName(r) = "Integer" Then
          v = v + r
      End If
   Next
   ReturnSumm = v
End Function
```

Использование массива в качестве параметра функции

В VBA допустимо использование массива в качестве параметра процедуры. В этом случае, массив, используемый в качестве параметра, при описании процедуры надо объявить как динамический. В следующем примере (листинг 6.14) процедура SumArray возвращает сумму элементов массива, который является ее первым параметром.

Листинг 6.14. Использование массива в качестве параметра функции

```
Function SumArray(ByRef A() As Double) As Double
   Dim x As Variant
   Dim s As Double
```

```
   s = 0
   For Each x In A
      s = s + x
   Next
   SumArray = s
End Function

Sub DemoSumArray()
   Dim s As Double
   Dim B(2, 2) As Double
   B(1, 1) = 1: B(1, 2) = 5
   B(2, 1) = 4: B(2, 2) = 5
   MsgBox SumArray(B)
End Sub
```

Функция, возвращающая массив значений

Часто при работе с таблицами возникает необходимость применить одну и ту же операцию к целому диапазону ячеек или произвести расчеты по формулам, зависящим от большого массива данных. Под массивом в MS Excel понимается прямоугольный диапазон формул или значений, которые MS Excel обрабатывает как единую группу. Для этих целей MS Excel предоставляет простое и элегантное средство — формулы массива. Продемонстрируем их создание на примере простой функции, возвращающей массив названий целых чисел (листинг 6.15). В этой функции, прежде чем выводить данные, надо проверить ориентацию массива, а именно, отображаются ли данные вдоль строки или столбца. Для этого достаточно подсчитать число строк выделенной области. Если оно больше единицы, то вдоль столбца, а в противном случае — вдоль строки.

Листинг 6.15. Функция, возвращающая массив значений

```
Function NameOfNumbers() As Variant
   Dim DataRows(1, 0) As String
   DataRows(0, 0) = "One"
   DataRows(1, 0) = "Two"
   Dim DataCols(0, 1) As String
   DataCols(0, 0) = "One"
   DataCols(0, 1) = "Two"
   If Selection.Rows.Count > 1 Then
      NameOfNumbers = DataRows
```

```
   Else
      NameOfNumbers = DataCols
   End If
End Function
```

Для применения данной функции:

1. Выберите диапазон, например `C2:C3`, в котором разместите массив названий чисел. От диапазона, в котором будет размещен результат, требуется, чтобы он имел искомый размер.
2. Введите формулу

   ```
   =NameOfNumbers()
   ```

3. Завершите ввод формулы не нажатием клавиши <Enter>, а нажатием комбинации клавиш <Ctrl>+<Shift>+<Enter>. Таким образом, вы сообщите MS Excel, что необходимо выполнить операцию над массивом, т. е. создать формулу массива. В ответ MS Excel автоматически возьмет формулу в фигурные скобки для того, чтобы информировать пользователя, что это не простая формула, а формула массива (рис. 6.3):

   ```
   {=NameOfNumbers()}
   ```

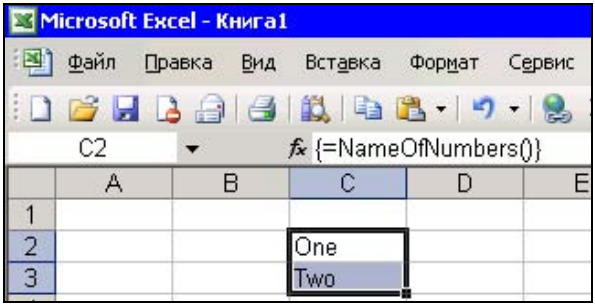


Рис. 6.3. Работа с функцией, возвращающей массив

Функция, вид которой зависит от параметра

В зависимости от бизнес-логики приложения можно конструировать функцию, содержание которой может видоизменяться. Например, необходимо сконструировать приложение, в котором пользователь может выбрать функцию проведения статистических расчетов, например суммарное и среднее значение выборки данных. Конечно, в этой ситуации можно воспользоваться двумя функциями, а можно и ограничиться только одной, как показано в листинге 6.16, у которой имеются два параметра: первый задает ссылку на диапазон, а второй — тип расчетов.

Листинг 6.16. Функция, вид которой зависит от параметра

```
Function StatFunction(rgn As Range, op As String) As Double
   Select Case LCase(op)
      Case "sum"
         StatFunction = Application.WorksheetFunction.Sum(rgn)
      Case "average"
         StatFunction = Application.WorksheetFunction.Average(rgn)
   End Select
End Function
```

Расчет следующего понедельника

В качестве примера конструирования пользовательских функций по работам с датами построим функцию, возвращающую следующий понедельник по отношению к указанной дате (листинг 6.17). В этой функции ключевую роль играет функция `Weekday`, возвращающая порядковый номер дня недели специфицированной даты. Второй параметр этой функции, являющийся необязательным, устанавливает начальный день недели.

Листинг 6.17. Расчет следующего понедельника

```
Function NextMonday(d As Date) As Date
   If Weekday(d, vbMonday) = 1 Then
      NextMonday = d
   Else
      NextMonday = d + 8 - Weekday(d, vbMonday)
   End If
End Function
```

Например, для определения первого понедельника после 12 сентября 2003 г. надо воспользоваться на рабочем листе следующей формулой:

```
=NextMonday(ДАТА(2003;9;12))
```

Определение возраста человека

Для определения возраста человека можно воспользоваться функцией из листинга 6.18. Ключевой здесь является функция `DateDiff` — возрастающая разность между двумя датами. Первый ее параметр, задающий единицу измерения, имеет следующие допустимые значения: `yyyy` (год), `q` (квартал), `m` (месяц), `d` (день), `w` (выходные), `ww` (неделя), `h` (час), `n` (минута), `s` (секунда).

Второй параметр функции задает начальную, а третий — конечную даты, которая в данном случае является текущей и возвращается функцией Now.

Листинг 6.18. Определение возраста человека

```
Function Age(DateOfBirth As Date) As Integer
   Age = DateDiff("yyyy", DateOfBirth, Now)
End Function
```

Надстройки

При создании функций, достаточно часто используемых, их рекомендуется разместить в файлах надстройки, MS Excel файлах с расширением xla. Основным преимуществом подобной операции является то, что настройки позволяют использовать функции в любой рабочей книге без ее спецификации. Например, если функция SinPi расположена в книге FunCollection.xsl, то в другой рабочей книге на нее надо ссылаться следующим образом:

```
=FunCollection.xsl!SinPi(A1)
```

если эта функция расположена в надстройке, то на нее можно ссылаться только по имени:

```
=SinPi(A1)
```

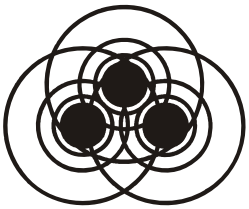
Для создания надстройки:

- расположите функции в модуле рабочей книги;
- в редакторе VBA выберите команду **Tools | VBAProject Properties**. На вкладке **Protection** окна **VBAProject — Project Properties** установите флажок **Lock project for viewing** и в поля **Password** и **Confirm password** введите и подтвердите пароль. Нажмите кнопку **OK**. Теперь без знания пароля никто не сможет просматривать ваш код и вносить в него изменения;
- выберите команду **Файл | Сохранить как** и в появившемся окне **Сохранение документа** в списке **Тип файла** выберите **Надстройка Microsoft Excel**. В поле **Имя файла**, в списке **Папка** специфицируете каталог расположения надстройки. Нажмите кнопку **Сохранить**. Рабочая книга будет сохранена как XLA файл.

Установить надстройку можно следующим образом:

- выберите команду **Сервис | Надстройка**;
- в появившемся диалоговом окне **Надстройка** при помощи кнопки **Обзор** добавьте ссылку на искомый XLA файл и нажмите кнопку **OK**.

Глава 7



VBA и Excel

В данной главе вы познакомитесь с объектной моделью Excel. Узнаете об ее основных объектах `Application`, `Workbook`, `Worksheet`, `Range` и `Selection`. Познакомитесь со свойствами, методами и событиями, связанными с этими и другими базовыми объектами Excel. Узнаете, как отсылается электронная почта, как можно вводить в диапазон неповторяющиеся значения, как установить выполнение специфицированной процедуры при нажатии заданной комбинации клавиш, как создать бегущую строку или картинку, как производить поиск файлов, как переопределить горячие клавиши приложения, как озвучить текст, как произвести условное форматирование, как управлять уровнем безопасности, как отображать результат, только при условии ввода всех данных, как проверять вводимые значения.

Объектная модель

Объектная модель MS Excel представляет собой иерархию объектов, подчиненных одному объекту `Application`, который соответствует самому приложению Excel. Многие из этих объектов собраны в библиотеке объектов Excel, но некоторые из них, например, объект `Assistant`, входят в библиотеку объектов Office, которая является общей для всех офисных приложений. На рис. 7.1 собраны основные компоненты объектной модели MS Excel. На этом рисунке имена семейств заканчиваются буквой "s", подчеркивая множественность по правилам английского языка. Каждый из объектов будет обсужден по ходу рассмотрения в книге, поэтому в данный момент ограничимся только представлением общей схемы объектной модели.

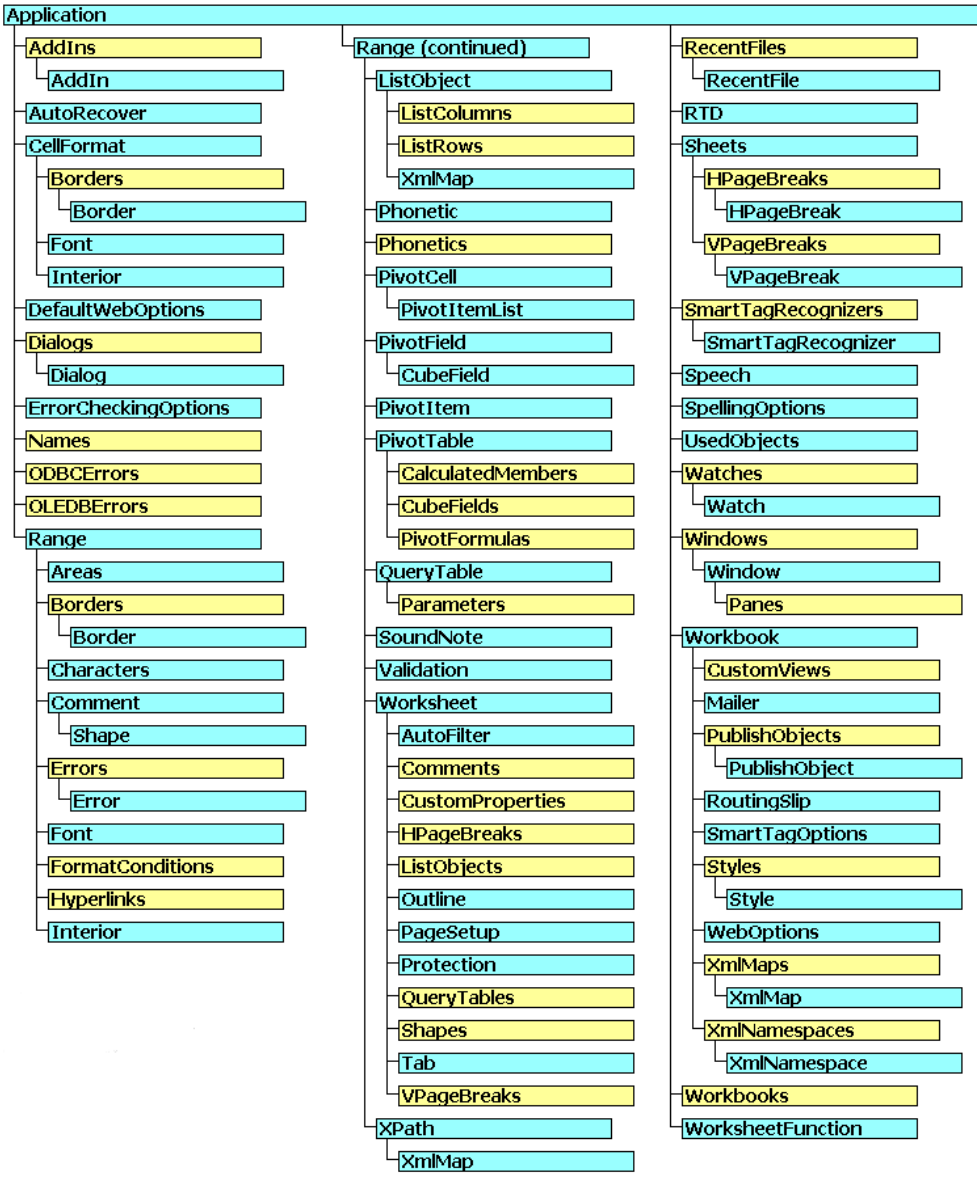


Рис. 7.1. Объектная модель MS Excel

Полная и неявная ссылка на объект

Полная ссылка на объект состоит из ряда имен вложенных последовательно друг в друга объектов. Разделителями имен объектов в этом ряду являются

точки, ряд начинается с объекта `Application` и заканчивается именем самого объекта. Например, полная ссылка на ячейку **A1** рабочего листа **Продажи** рабочей книги с именем **Архив** имеет вид:

```
Application.Workbooks("Архив").Worksheets("Продажи").Range("A1")
```

Приводить каждый раз полную ссылку на объект совершенно не обязательно. Обычно достаточно ограничиться только неявной ссылкой на объект. В неявной ссылке, в отличие от полной, объекты, которые активны в данный момент, как правило, можно опускать. В рассмотренном случае, если ссылка на ячейку **A1** дана в программе, выполняемой в среде Excel, то ссылка на объект `Application` может быть опущена, т. е. достаточно привести относительную ссылку:

```
Workbooks("Архив").Worksheets("Продажи").Range("A1")
```

Если в этом примере ссылки рабочая книга **Архив** является активной, то ссылку можно еще сократить:

```
Worksheets("Продажи").Range("A1")
```

Если и рабочий лист **Продажи** активен, то в относительной ссылке вполне достаточно ограничиться упоминанием только диапазона **A1**:

```
Range("A1")
```

Объект *Application*

Объект `Application` — это главный (корневой) объект в иерархии объектов MS Excel, представляющий само приложение MS Excel. Он имеет огромное число свойств и методов, которые позволяют установить общие параметры приложения MS Excel.

Свойства объекта *Application*

Объект `Application`, благодаря обширной коллекции свойств, позволяет программно установить значения многих опций окна **Параметры**, отображаемого при выборе команды **Сервис | Параметры**. Кроме того, он обеспечивает доступ к объектам верхнего уровня типа `ActiveCell`, `ActiveSheet` и т. д. Перечислим основные свойства этого объекта.

```
ActiveCell                      ActiveChart
ActivePrinter                   ActiveSheet
ActiveWorkbook                  AddIns
AlertBeforeOverwriting          Assistant
AutoCorrect                     AutoFormatAsYouTypeReplaceHyperlinks
AutomationSecurity              AutoRecover
CalculateBeforeSave             Calculation
CalculationState                Caption
```

```
CellDragAndDrop                Cells
Charts                         Columns
Cursor                         DataEntryMode
DefaultFilePath                DefaultSaveFormat
DefaultWebOptions              Dialogs
DisplayAlerts                  DisplayCommentIndicator
DisplayFormulaBar              DisplayFullScreen
DisplayScrollBars              DisplayStatusBar
EditDirectlyInCell             EnableCancelKey
FileSearch                     FoundFiles
Height                         Left
LibraryPath                    MemoryFree
MemoryTotal                    MemoryUsed
Names                          OperatingSystem
OrganizationName               Path
ProductCode                    RecentFiles
ReferenceStyle                 ScreenUpdating
Selection                      Sheets
StatusBar                      ThisWorkbook
Top                            UsableHeight
UsableWidth                    UserName
Version                        Width
Windows                        WindowState
WorksheetFunction
```

Ссылка на активную рабочую книгу, лист, ячейку, диаграмму и принтер

Свойства `ActiveWorkbook`, `ActiveSheet`, `ActiveCell`, `ActiveChart`, `ActivePrinter` объекта `Application` возвращают активную рабочую книгу, лист, ячейку, диаграмму и принтер. Объект `ActiveCell` содержится в `ActiveSheet`, а объекты `ActiveSheet` и `ActiveChart` в `ActiveWorkbook`. В следующем примере (листинг 7.1) в активной ячейке устанавливается красный цвет фона с зеленым полужирным шрифтом и в нее вводится строка текста `Май`. Активный рабочий лист переименовывается в `Отчет за май`, а цвет его ярлыка назначается желтым.

Листинг 7.1. Ссылка на активный рабочий лист и ячейку

```
Sub DemoActives()
   With ActiveSheet
      .Tab.ColorIndex = 27 ' Желтый цвет
      .Name = "Отчет за май"
```

```
   End With
   With ActiveCell
      .Font.Bold = True
      .Font.Color = vbGreen
      ' vbGreen — встроенная константа, задающая зеленый цвет
      .Value = "Май"
      .Interior.Color = vbRed
       ' vbRed — встроенная константа, задающая красный цвет
   End With
End Sub
```

Загрузка инсталлированной надстройки

Свойство `AddIns` объекта `Application` возвращает семейство инсталлированных надстроек. Например, в следующем коде (листинг 7.2) функция `LoadAddIn` проверяет, установлена ли специфицированная надстройка (в данном случае **Solver Add-in**). Эта функция возвращает значение:

- 0, если надстройка загружена;
- 1, если надстройка установлена, но загрузить ее пришлось при помощи данной функции;
- −1, если надстройка не найдена;
- −2, если произошла ошибка при работе функции.

Напомним, что надстройку можно инсталлировать, выбрав команду **Сервис | Надстройки**.

Листинг 7.2. Проверка, установлена ли указанная надстройка

```
Function LoadAddIn(AddInTitle) As Integer
   Dim ads As AddIn
   On Error GoTo ErrorHandler
   For Each ads In Application.AddIns
      If ads.Title = AddInTitle Then
         LoadAddIn = 0
         If Not ads.Installed Then
            AddIns(AddInTitle).Installed = True
            LoadAddIn = 1
         End If
         Exit Function
      End If
```

```
    Next
    LoadAddIn = -1
    Exit Function
ErrorHandler:
    LoadAddIn = -2
End Function

Sub Test()
  MsgBox LoadAddIn("Solver Add-in")
End Sub
```

Управление уровнем безопасности

Свойство `AutomationSecurity` объекта `Application` позволяет управлять уровнем безопасности при открытии рабочей книги. Это свойство возвращает одну из следующих `MsoAutomationSecurity` констант:

- `msoAutomationSecurityByUI` — используется уровень безопасности, специфицированный диалоговым окном, отображаемым при открытии рабочей книги;
- `msoAutomationSecurityForceDisable` — отключает без предупреждающего сообщения все макросы в открываемых рабочих книгах;
- `msoAutomationSecurityLow` — все макросы включены.

Следующая процедура (листинг 7.3) позволяет, используя диалоговое окно **Open**, открыть рабочую книгу с автоматическим отключением макросов. Для вызова диалогового окна **Open** применено свойство `FileDialog` объекта `Application`.

Листинг 7.3. Открытие рабочей книги с автоматическим отключением макросов

```
Sub OpenWithMacrosOff()
    Dim secAutomation As MsoAutomationSecurity
    secAutomation = Application.AutomationSecurity
    Application.AutomationSecurity = msoAutomationSecurityForceDisable
    Application.FileDialog(msoFileDialogOpen).Show
    Dim fn As String
    fn = Application.FileDialog(msoFileDialogFilePicker).SelectedItems(1)
    Application.Workbooks.Open fn
    Application.AutomationSecurity = secAutomation
End Sub
```

Рабочая книга, в которой выполняется данный макрос

Свойство `ThisWorkbook` объекта `Application` возвращает рабочую книгу, содержащую выполняющийся в данный момент макрос. Это свойство может возвращать рабочую книгу, отличную от книги, возвращаемой свойством `ActiveWorkbook`, т. к. выполняемый макрос может находиться в неактивной книге.

Установка заголовка окна MS Excel

Свойство `Caption` объекта `Application` возвращает или устанавливает текст из заголовка главного окна MS Excel. Установка значения свойства равным `Empty` возвращает заголовок, используемый по умолчанию. В следующем примере (листинг 7.4) первая процедура обрабатывает событие `Open` рабочей книги и устанавливает в качестве заголовка окна приложения текст `Отчет за 2003 год`, а вторая — обрабатывает событие `BeforeClose` рабочей книги и возвращает окну имя, используемое по умолчанию, т. е. `Microsoft Excel` (рис. 7.2).

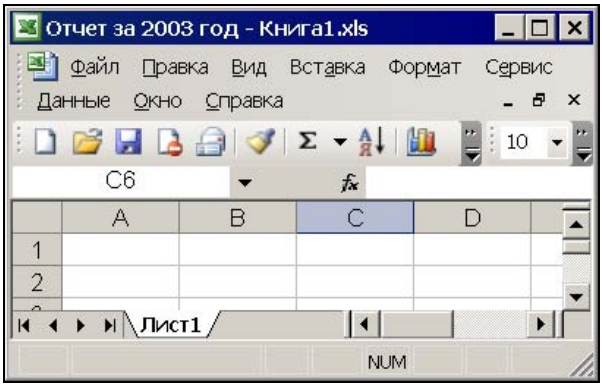


Рис. 7.2. Окно MS Excel с пользовательским заголовком

Листинг 7.4. Установка заголовка окна MS Excel. Модуль `ЭтаКнига`

```
Private Sub Workbook_Open()
   Application.Caption = "Отчет за 2003 год"
End Sub

Private Sub Workbook_BeforeClose(Cancel As Boolean)
   Application.Caption = Empty
End Sub
```

Работа со строкой состояния

Свойство `StatusBar` объекта `Application` задает текст, отображаемый в строке состояния приложения. Если значение этого свойства равно `False`, то отображается текст, заданный по умолчанию. В следующем примере (листинг 7.5) конструируется бегущая строка состояния. Эффект бега создается постепенным изменением количества пробелов, размещенных перед отображаемым текстом. Метод `Wait` объекта `Application` обеспечивает задержку выполнения приложения на указанный промежуток времени, в данном случае секунду.

Листинг 7.5. Бегущий текст в строке состояния

```
Function StatusString(ByVal LeftSpace As Integer, _
                      ByVal Message As String) As String
   StatusString = Space(LeftSpace) & Message
End Function

Sub RunMessage()
   Dim n As Integer
   For n = 20 To 0 Step -1
      Application.StatusBar = StatusString(n, "Hello!")
      Application.Wait Now() + TimeValue("00:00:01")
      DoEvents
   Next
   Application.StatusBar = False
End Sub
```

Установка выполнения специфицированной процедуры при нажатии заданной комбинации клавиш

Метод `OnKey` устанавливает выполнение специфицированной процедуры при нажатии заданной комбинации клавиш.

`OnKey(Key, Procedure)`, где:

- *`Procedure`* — необязательный параметр, задающий имя процедуры, выполняемой при нажатии комбинации клавиш, указанных в качестве значения параметра *`Key`*. Если параметр опущен, то выполняется действие, которое в системе зарезервировано за этой комбинацией клавиш;
- *`Key`* — обязательный параметр, задающий строку, определяющую комбинацию клавиш, которая должна быть нажата. В этой строке можно также указывать специальные клавиши, используя коды, приведенные в табл. 7.1. Допустимо использование сочетания одновременно нажатой специальной

клавиши и клавиши модификатора. С этой целью установлены коды клавиш модификаторов, перечисленные в табл. 7.2.

Таблица 7.1. *Коды специальных клавиш*

Специальная клавиша	Код
<Backspace>	{BACKSPACE} **или** {BS}
<Break>	{BREAK}
<Caps Lock>	{CAPSLOCK}
<Delete>	{DELETE} **или** {DEL}
< >	{DOWN}
<End>	{END}
<Enter> (числовая клавиатура)	{ENTER}
<Enter>	~
<Esc>	{ESCAPE} **или**{ESC}
<Home>	{HOME}
<Insert>	{INSERT}
< >	{LEFT}
<Num Lock>	{NUMLOCK}
<Page Down>	{PGDN}
<Page Up>	{PGUP}
<Return>	{RETURN}
< >	{RIGHT}
<Scroll Lock>	{SCROLLLOCK}
<Tab>	{TAB}
< >	{UP}
От <F1> до <F15>	От {F1} до {F15}

Таблица 7.2. *Коды клавиш модификаторов*

Клавиша модификатор	Код
<Shift>	+
<Ctrl>	^
<Alt>	%

В следующем примере процедуре `Амортизация` назначена комбинация клавиш <Ctrl>+<+>, а процедуре `Ставка` — <Shift>+<Ctrl>+<→>

```
Application.OnKey "^{+}", "Амортизация"
Application.OnKey "+^{RIGHT}", "Ставка"
```

Как переопределить горячие клавиши приложения

Метод `OnKey` может с успехом применяться для переопределения горячих клавиш, присущих данному приложению. При этом его иногда применяют в комбинации с оператором SendKeys, который посылает последовательность символов активному окну, как если бы они были набраны на клавиатуре.

Например, в следующем приложении (листинги 7.6, *а*, *б*) при нажатии комбинации клавиш <Ctrl>+<o> вместо окна **Открытие документа** отображается окно с сообщением **Это заглушка!** При нажатии комбинации клавиш <Ctrl>+<x> вместо операции вставки из буфера обмена в книгу добавится новый рабочий лист. А при нажатии комбинации клавиш <Ctrl>+<p> вместо печати ничего не происходит.

Листинг 7.6, *а*. Переопределение горячих клавиш приложения. Модуля `ЭтаКнига`

```
Private Sub Workbook_Open()
   With Application
      .OnKey "^o", "SetDummy"
      .OnKey "^x", "SetNewAction"
      .OnKey "^p", ""
   End With
End Sub
```

Листинг 7.6, *б*. Переопределение горячих клавиш приложения. Стандартный модуль

```
Public Sub SetDummy()
   MsgBox "Это заглушка!"
End Sub
Public Sub SetNewAction()
   SendKeys "^n"
End Sub
```

Установка курсора

Свойство `Cursor` объекта `Application` устанавливает тип курсора в приложении. Допустимые значения:

- `xlDefault` (курсор, используемый по умолчанию);

- `xlWait` (песочные часы);
- `xlNorthwestArrow` (стрелка в северо-западном направлении);
- `xlIBeam` (вертикальная полоска).

Если вы используете нестандартный курсор, например, для уведомления пользователя, что приложение не зависло, а идут вычисления, то по завершению вычислений не забудьте установить курсор, используемый по умолчанию в системе. Например, следующий код (листинг 7.7) находит сумму обратных величин квадратов целых чисел. Так как суммируется достаточно много чисел, на время суммирования курсор принимает вид песочных часов. По завершению суммирования курсору назначается вид, используемый по умолчанию.

Листинг 7.7. Изменение курсора на время вычислений

```
Sub DemoCursor()
   Dim s As Double
   Dim i As Long
   Application.Cursor = xlWait
   s = 0
   For i = 1 To 1000000
      s = s + 1 / i ^ 2
   Next
   Application.Cursor = xlDefault
   MsgBox s
End Sub
```

Видоизменять курсор можно не только при вычислениях, но и при любых других действиях, например при выборе ячейки. В следующем примере при выборе ячейки **A1** курсор превращается в указатель, направленный на северо-запад, а при выборе любой другой он принимает вид, который используется в приложении по умолчанию (листинг 7.8).

Листинг 7.8. Изменение курсора после выбора специфицированной ячейки. Модуль рабочего листа

```
Private Sub Worksheet_SelectionChange(ByVal Target As Range)
   If Target.Address = "$A$1" Then
      Application.Cursor = xlNorthwestArrow
   Else
     Application.Cursor = xlDefault
   End If
End Sub
```

Семейство встроенных диалоговых окон

Свойство `Dialogs` возвращает семейство `Dialogs` всех встроенных диалоговых окон. Параметр этого семейства идентифицирует окно. Метод `Show` отображает его на экране, а параметры этого метода задают параметры, специфицируемые в отображаемом окне. Метод `Show` возвращает значение `True`, если задача, поставленная в отображаемом окне, была выполнена успешно. Например, следующий код (листинг 7.9) отображает окно **Открытие документа** для открытия книги C:\test.xls.

Листинг 7.9. Открытие книги при помощи встроенного окна

```
Sub DemoDialogs()
   Dim idx As Long
   idx = Application.Dialogs(xlDialogOpen).Show("c:\test.xls")
   If idx Then
      MsgBox "Файл открыт"
   Else
      MsgBox "Файл не открыт"
   End If
End Sub
```

Объект *FileDialog*

Объект `FileDialog`, возвращаемый свойством `FileDialog` объекта `Application`, предоставляет в распоряжение разработчика диалоговые окна **Открыть** и **Сохранить как**. Свойство `FileDialog` имеет один параметр `DialogType`, задающий тип окна. У этого параметра имеются четыре допустимых значения:

- `msoFileDialogFilePicker` — позволяет пользователю выбрать файл;
- `msoFileDialogFolderPicker` — позволяет пользователю выбрать папку;
- `msoFileDialogOpen` — позволяет пользователю открыть выбранный файл. Открытие файла производится методом `Execute`;
- `msoFileDialogSaveAs` — позволяет пользователю сохранить файл. Сохранение файла производится методом `Execute`.

Для отображения окна, симулируемого объектом `FileDialog`, необходимо воспользоваться методом `Show`. Этот метод возвращает значение 0, если нажата кнопка **Отмена** и значение −1, если нажата другая функциональная кнопка. Для окон **Открыть** и **Сохранить как** после применения метода `Show` надо воспользоваться методом `Execute` для реализации выбранной команды.

В следующих трех примерах демонстрируется техника сохранения и загрузки файлов при помощи окон, имеющих типы `msoFileDialogFilePicker`

(листинг 7.10), `msoFileDialogOpen` (листинг 7.11) и `msoFileDialogSaveAs` (листинг 7.12).

Листинг 7.10. Загрузка файлов с помощью окна `msoFileDialogFilePicker`

```
Sub LoadFiles()
   Dim fd As FileDialog
   Set fd = Application.FileDialog(msoFileDialogFilePicker)
   Dim itm As Variant
   With fd
      If .Show = -1 Then
         For Each itm In .SelectedItems
            Workbooks.Add itm
         Next
      End If
   End With
   Set fd = Nothing
End Sub
```

Листинг 7.11. Загрузка файла с помощью окна `msoFileDialogOpen`

```
Sub LoadFile()
   Dim fd As FileDialog
   Set fd = Application.FileDialog(msoFileDialogOpen)
   If fd.Show = -1 Then
      fd.Execute
   Else
      MsgBox "Выбрали отмену"
   End If
   Set fd = Nothing
End Sub
```

Листинг 7.12. Сохранение файла с помощью окна `msoFileDialogSaveAs`

```
Sub SaveFile()
   Dim fd As FileDialog
   Set fd = Application.FileDialog(msoFileDialogSaveAs)
   If fd.Show = -1 Then
      fd.Execute
```

```
   Else
      MsgBox "Выбрали отмену"
   End If
   Set fd = Nothing
End Sub
```

Отображение встроенных предупреждений о работе программы

Свойство `DisplayAlerts` объекта `Application` устанавливает, надо ли отображать встроенные предупреждения о работе программы. Например, следующая процедура (листинг 7.13), обрабатывающая событие `Open` объекта `Workbook`, отменяет отображение предупреждений при работе с книгой.

Листинг 7.13. Отмена отображения предупреждения о необходимости сохранять документ

```
Private Sub Workbook_Open()
   Application.DisplayAlerts = False
End Sub
```

Поиск файлов

Свойство `FileSearch` объекта `Application` возвращает объект `FileSearch`, который инкапсулирует в себе свойства и методы, реализующие поиск специфицированного файла на диске. Перечислим основные свойства объекта `FileSearch`:

- свойство `LookIn` возвращает или устанавливает каталог, в котором производится поиск;
- свойство `FileType` возвращает или устанавливает тип искомого файла. Его допустимым значением может быть одна из следующих `MsoFileType` констант:

```
msoFileTypeAllFiles                    msoFileTypeBinders
msoFileTypeCalendarItem                msoFileTypeContactItem
msoFileTypeCustom                      msoFileTypeDatabases
msoFileTypeDataConnectionFiles         msoFileTypeDesignerFiles
msoFileTypeDocumentImagingFiles        msoFileTypeExcelWorkbooks
msoFileTypeJournalItem                 msoFileTypeMailItem
msoFileTypeNoteItem                    msoFileTypeOfficeFiles
msoFileTypeOutlookItems                msoFileTypePhotoDrawFiles
msoFileTypePowerPointPresentations     msoFileTypeProjectFiles
msoFileTypePublisherFiles              msoFileTypeTaskItem
msoFileTypeTemplates                   msoFileTypeVisioFiles
msoFileTypeWebPages                    msoFileTypeWordDocuments
```

- свойство `FoundFiles` возвращает объект `FoundFiles`, представляющий собой список имен всех найденных в течение поиска файлов.

Метод `Execute` объекта `Application` производит непосредственный поиск. Он возвращает целое число, причем, если оно равно 0, то ни одного файла не было найдено, а если положительное число, то найден, по крайней мере, один файл:

```
Execute(SortBy, SortOrder, AlwaysAccurate)
```

Где:

- *`SortBy`* — необязательный параметр, определяющий критерий сортировки найденных файлов. Допустимые значения определяются следующими `MsoSortBy` константами:

  `msoSortByFileName` `msoSortByFileType` `msoSortByLastModified` `MsoSortByNone` `MsoSortBySize`;

- *`SortOrder`* — необязательный параметр, определяющий порядок сортировки. Допустимые значения определяются следующими `MsoSortOrder` константами: `msoSortOrderAscending`, `msoSortOrderDescending`;

- *`AlwaysAccurate`* — необязательный параметр, принимающий логические значения и определяющий, должны ли при поиске просматриваться те файлы, которые были добавлены, модифицированы или удалены после последнего изменения файлового индекса.

В приводимом ниже примере (листинг 7.14) производится поиск всех рабочих книг в корневом каталоге диска **C**. Если таковые имеются, то отображается сообщение с указанием их количества и полным списком имен. Если же их нет, то пользователь также об этом информируется.

Листинг 7.14. Поиск рабочих книг в корневом каталоге диска C:

```
With Application.FileSearch
   .LookIn = "c:\"
   .FileType = msoFileTypeExcelWorkbooks
   If .Execute(SortBy:=msoSortByFileName, _
               SortOrder:=msoSortOrderAscending) > 0 Then
      Dim str As String
      str = "Найдено " & .FoundFiles.Count & " книг(и):" & vbCr
      Dim i As Integer
      For i = 1 To .FoundFiles.Count
         str = str & .FoundFiles(i) & vbCr
      Next
      MsgBox str
```

```
   Else
      MsgBox "Рабочие книги не найдены."
   End If
End With
```

Печать активного листа

Печать активного листа рабочей книги производится методом `PrintOut`. Для того чтобы получить ссылку на активный лист, воспользуйтесь свойством `ActiveSheet` объекта `Application`.

```
ActiveSheet.PrintOut
```

Предварительный просмотр книги

Предварительный просмотр рабочей книги реализуется методом `PrintPreview` семейства `Sheets`. Если у свойства `Sheets` объекта `Application` не указывается значение индекса, то производится просмотр всей книги.

```
Application.Sheets.PrintPreview
```

Если же у свойства `Sheets` индекс специфицирован, то осуществляется просмотр указанного листа, например в следующей инструкции первого.

```
Application.Sheets(1).PrintPreview
```

Доступ из кода к функциям рабочего листа

Объект `Application` через свойство `WorksheetFunction` предоставляет возможность использовать в коде все встроенные функции рабочего листа. Это свойство возвращает `WorksheetFunction` объект, являющийся контейнером всех функций рабочего листа. Например, в следующей инструкции переменной `MaxValue` присваивается максимальное значение из диапазона **A1:A4**:

```
Dim MaxValue As Double
MaxValue = Application.WorksheetFunction.Max(Range("A1:A4"))
```

Примечание

Функции рабочего листа можно включать в код непосредственно через объект `Application`, опуская свойство `WorksheetFunction`. Например, в следующем примере переменной `Pi` присваивается значение π, а переменной `Сумма` присваивается сумма значений из диапазона **A1:A4**:

```
Pi = Application.Pi()
Сумма = Application.Sum(Range("A1:A4"))
```

Но хотя приводимые инструкции короче, их применение не всегда удобно. Дело в том, что после ввода в модуле `Application.WorksheetFunction` на экране отображается список с функциями рабочего листа, в то время как после ввода `Application` список с функциями рабочего листа не появляется. Поэтому, если вы точно знаете имя вводимой функции, то второй способ вполне приемлем. Если же вы не уверены в имени, то у первого способа нет альтернативы.

Номер версии *MS Excel*

Свойство `Version` объекта `Application` возвращает номер текущей версии MS Excel. Используется для проверки, применяется ли корректная версия. Например, следующая инструкция проверяет версию MS Excel, и если она отлична от 11.0, то происходит выход из процедуры.

```
If Application.Version <> "11.0" Then Exit Sub
```

Определение локальной версии Excel и установок Windows

Определить локальную версию Excel и установки Windows можно при помощи свойства `International` объекта `Application`, например, как показано в листинге 7.15. Если значение параметра этого свойства равно `xlCountrySetting`, то идентифицируется локализация установок Windows, а если равно `xlCountryCode`, то Excel.

Листинг 7.15. Определение локальной версии Excel и установок Windows

```
Sub DemoCheck()
   CheckInternational Application.International(xlCountrySetting), _
                      "Версия Windows"
   CheckInternational Application.International(xlCountryCode), _
                      "Версия Office"
End Sub

Sub CheckInternational(n As Integer, msg As String)
   Select Case n
      Case 1
         msg = msg + " Английская (США)"
      Case 7
         msg = msg + " Русская"
      Case 49
         msg = msg + " Немецкая (Германия)"
      Case Else
         msg = msg + " не учтена"
   End Select
   MsgBox msg
End Sub
```

Методы объекта *Application*

У объекта `Application` имеется большая коллекция методов, позволяющих производить различные действия — от конвертации метрических единиц измерения до создания таймера. Перечислим основные методы этого объекта:

`ActivateMicrosoftApp`
`CentimetersToPoints`
`ConvertFormula`
`Help`
`InputBox`
`OnKey`
`Quit`
`Save`
`Volatile`
`Calculate`
`CheckSpelling`
`Evaluate`
`InchesToPoints`
`Intersect`
`OnTime`
`Run`
`Union`
`Wait`

Проверка правописания отдельного слова

Проверку правописания можно осуществить методом `CheckSpelling` объекта `Application`, который возвращает значение `True`, если специфицированное слово находится в словаре и значение `False` в противном случае.

`CheckSpelling(`*`Word`*`, `*`CustomDictionary`*`, `*`IgnoreUppercase`*`)`, где:

- *`Word`* — обязательный параметр, используемый только с объектом `Application`, специфицирующий проверяемое слово;
- *`CustomDictionary`* — необязательный параметр, указывающий имя файла с пользовательским словарем;
- *`IgnoreUppercase`* — необязательный параметр, принимающий логические значения и определяющий, надо ли проверять правописание слов, набранных строчными буквами.

Например, следующий код (листинг 7.16) проверяет принадлежность указанного слова словарю.

Листинг 7.16. Проверка принадлежности указанного слова словарю

```
Dim wrd As String
wrd = "Привет"
If Application.CheckSpelling(wrd) Then
   MsgBox "Слово '" & wrd & "' найдено"
Else
   MsgBox "Слово '" & wrd & "' не найдено"
End If
```

Преобразование имени MS Excel в объект или значение

Метод `Evaluate` объекта преобразует имя MS Excel в объект или значение.

```
Evaluate(Name)
```

Здесь *Name* — обязательный параметр типа `String`, задающий конвертируемое имя. Допустимы следующие типы имен:

- ссылка формата **A1** на ячейку, которая рассматривается как абсолютная ссылка;
- диапазон;
- пользовательское имя;
- внешние ссылки. При этом надо использовать оператор ! для указания ссылки на объект из другой книги. Например, `Evaluate("[Отчет.XLS]Май!A1")`.

Симулирование вычисления арифметических выражений

Метод `Evaluate` позволяет по строковому представлению арифметического выражения находить его значение. Например, следующий оператор (листинг 7.17) отображает значение выражения `(1+3)/2+sin(2)`.

Листинг 7.17. Калькулятор

```
MsgBox Application.Evaluate("(1+3)/2+sin(2)")
```

Простейший функциональный калькулятор

Обобщая идею, обсужденную в предыдущем разделе, используя метод `Evaluate`, легко создать простейший функциональный калькулятор, который возвращает значение введенного арифметического выражения. Для этого надо иметь средство ввода пользователем строкового представления арифметического выражения, что можно обеспечить методом `InputBox`. В следующем коде (листинг 7.18) метод `Evaluate` произведет требуемые вычисления, а диалоговое окно `MsgBox` обеспечит отображения результата.

Листинг 7.18. Простейший функциональный калькулятор

```
Sub Calculator()
   Dim s As String
   s = InputBox("Введите арифметическое выражение")
   MsgBox s & " = " & Application.Evaluate(s)
End Sub
```

Симулирование ячеек рабочего листа

Метод `Evaluate` позволяет симулировать работу с ячейками или диапазонами рабочего листа без реального воплощения этих действий на рабочем листе. Например, в следующем коде (листинг 7.19) симулируется ввод в ячейку **A1** значения `25`, ввод в ячейку **A2** формулы `=A1^2` и считывание возвращаемого этой формулой значения.

Листинг 7.19. Симулирование ввода данных в ячейки и считывания из них значений

```
Evaluate("A1").Value = 25
Evaluate("A2").Formula = "=A1^2"
MsgBox Evaluate("A2").Value
```

Ту же задачу можно решить в следующем коде (листинг 7.20), но здесь уже производится симулирование не операций ввода и вывода, а самих ячеек.

Листинг 7.20. Симулирование ячеек

```
Dim firstCell As Range
Dim secondCell As Range
Set firstCell = Evaluate("A1")
Set secondCell = Evaluate("A2")
firstCell.Value = 25
secondCell.Formula = "=A1^2"
MsgBox secondCell.Value
```

Применение квадратных скобок при симулировании операций на рабочем листе

Вместо метода `Evaluate` со строкой в качестве значения его параметра, можно использовать эту же строку, но заключенную в квадратные скобки. Запись оператора будет короче, а результат — тот же.

```
[a1].Value = 25
[a2].Formula = "=A1^2"
MsgBox [a2].Value
```

или

```
Dim s As Double
s = [(1+3)/2+sin(2)]
MsgBox s
```

Назначение выполнения процедуры на определенное время

Метод `OnTime` объекта `Application` назначает выполнение указанной процедуры на определенное время.

`OnTime(EarliestTime, Procedure, LatestTime, Schedule)`, где:

- *`EarliestTime`* — обязательный параметр типа `Variant`, задающий момент запуска процедуры;
- *`Procedure`* — обязательный параметр типа `String`, задающий имя процедуры;
- *`LatestTime`* — необязательный параметр типа `Variant`. Если на момент запуска процедуры MS Excel не может ее запустить в силу того, что он выполняет другое действие, то *`LatestTime`* определяет последнее время ее запуска. Если этот параметр опущен, то MS Excel будет ждать до тех пор, пока не сможет выполнить данную процедуру;
- *`Schedule`* — необязательный параметр типа `Variant`. Если его значение равно `True`, то указанная процедура установлена на выполнение в специфицированное время. Если же значение равно `False`, то задание снято с исполнения.

Электронные часы в ячейке рабочего листа

Метод `OnTime` позволяет создать электронные часы. Для этого достаточно рекурсивно вызывать процедуру, в которой считывается текущее время. Затем оно выводится в ячейку рабочего листа, найденное время увеличивается на секунду, и уже для вычисленного нового времени устанавливается рекурсивный вызов процедуры (листинг 7.21).

Листинг 7.21. Электронные часы в ячейке рабочего листа. Стандартный модуль

```
Sub DemoClock()
   DemoOnTime
End Sub


Sub DemoOnTime()
   Dim newHour, newMinute, newSecond, newTime
   Cells(1, 1).Value = Now
   newHour = Hour(Now)
   newMinute = Minute(Now)
   newSecond = Second(Now) + 1
   newTime = TimeSerial(newHour, newMinute, newSecond)
```

```
   Application.OnTime EarliestTime:=newTime, _
                      Procedure:="DemoOnTime"
End Sub
```

Электронный будильник

Метод `OnTime` обеспечивает простую реализацию будильника. Для этого в качестве значения первого его параметра достаточно указать момент времени, когда будильник должен прозвонить, а в качестве значения второго параметра — ссылку на процедуру, которая и реализует "звонок". Например, приводимый ниже код (листинг 7.22) обеспечивает отображение сообщения `Слишком поздно!!!` в 50 минут после полуночи.

Листинг 7.22. Электронный будильник

```
Sub DemoAlarm()
   Application.OnTime EarliestTime:=TimeValue("00:50:00"), _
                      Procedure:="AlarmClock"
End Sub

Sub AlarmClock()
   MsgBox "Слишком поздно!!!"
End Sub
```

Отмена задания у часов

Если значение параметра *`Schedule`* метода `OnTime` установлено равным `False`, то установленное при помощи параметров *`EarliestTime`* и *`Procedure`* задание снимается с выполнения. Например, следующий код (листинг 7.23) отменяет выполнение процедуры, исполняющей роль будильника из предыдущего раздела, при двойном щелчке на ячейке рабочего листа.

Листинг 7.23. Отмена задания у часов. Модуль рабочего листа

```
Private Sub Worksheet_BeforeDoubleClick(ByVal Target As Range, _
      Cancel As Boolean)
   On Error Resume Next
   Application.OnTime EarliestTime:=TimeValue("00:50:00"), _
                      Procedure:="AlarmClock", _
                      Schedule:=False
End Sub
```

Закрытие приложения

Метод `Quit` объекта `Application` закрывает приложение. Например, следующий код (листинг 7.24) производит закрытие приложения при двойном щелчке на ячейке рабочего листа без предупреждения пользователя о необходимости сохранить обновленную версию рабочей книги.

Листинг 7.24. Закрытие приложения при двойном щелчке на ячейке рабочего листа. Модуль рабочего листа

```
Private Sub Worksheet_BeforeDoubleClick(ByVal Target As Range, _
   Cancel As Boolean)
   Application.DisplayAlerts = False
   Application.Quit
End Sub
```

Сохранение изменений во всех рабочих книгах

Метод `Save` объекта `Application` сохраняет на диск изменения указанной рабочей книги. Следующий код (листинг 7.25), обрабатывающий событие `BeforeRightClick` объекта `Worksheet`, при щелчке правой кнопкой на ячейке **A1** обеспечивает сохранение всех рабочих книг, открытых в данном приложении, с последующим его закрытием.

Листинг 7.25. Сохранение изменений во всех рабочих книгах. Модуль рабочего листа

```
Private Sub Worksheet_BeforeRightClick(ByVal Target As Range, _
      Cancel As Boolean)
   If Target Is Range("A1") Then
      Dim wb As Workbook
      For Each wb In Application.Workbooks
         wb.Save
      Next
      Application.Quit
   End If
End Sub
```

Приостановка работы приложения до указанного времени

Метод `Wait` объекта `Application` приостанавливает работу приложения без остановки работы других программ до указанного момента времени. Возвращает значение `True`, если специфицированный момент времени наступил.

```
Wait(Time)
```

Здесь параметр *Time* определяет время, на которое предполагается возобновить работу приложения.

В следующем примере устанавливается остановка работы приложения на 17 часов:

```
Application.Wait "17:00:00"
```

В данном примере приложение прерывает выполнение до указанного момента времени и начинает работу с отображения сообщения:

```
Dim delay As Date
delay = #11:11:00 PM#
If Application.Wait(delay) Then
   MsgBox "Пора взятся за дело"
End If
```

В последнем коде выполнение приложения задерживается на 10 с.:

```
Dim delay As Date
delay = Now + TimeValue("00:00:10")
Application.Wait delay
```

События объекта *Application*

Прежде чем применить событие объекта `Application`, необходимо создать модуль класса и там объявить объект типа `Application`, причем при этом объявлении использовать ключевое слово `WithEvents`. Например, как сделано в следующем коде (листинг 7.26, *а*). Кроме того, переименуйте модуль класса из безликого `Class1`, например, в `MyAppWithEvent`.

Листинг 7.26 *а*. Объявление объекта типа `Application`. Модуль класса

```
Public WithEvents App As Application
```

После этого имя объекта `App` будет добавлено в список объектов, приводимый в списке **General** модуля класса. В списке **Declarations** приводятся ассоциированные с ним события. Например, следующий код (листинг 7.26, *б*) реализует обработку двух событий `WorkbookActivate` и `WorkbookDeactivate`, генерируемых при активизации и деактивизации рабочей книги.

Листинг 7.26 *б*. Процедуры, обрабатывающие события объекта типа `Application`. Модуль класса

```
Private Sub App_WorkbookActivate(ByVal Wb As Workbook)
   MsgBox "Книга " & Wb.Name & " активизирована"
```

```
End Sub

Private Sub App_WorkbookDeactivate(ByVal Wb As Workbook)
   MsgBox "Книга " & Wb.Name & " деактивизирована"
End Sub
```

Теперь, для того чтобы процедура, обрабатывающая событие, исполнялась, остается объявить объект типа `MyAppWithEvent`, например, как это делается в следующем коде (листинг 7.26, *в*).

Листинг 7.26 *в*. Объявление объекта типа `MyAppWithEvent`. Модуль `ЭтаКнига`

```
Dim a As New MyAppWithEvent

Private Sub Workbook_Open()
   Set a.App = Application
End Sub
```

В заключение в табл. 7.3 перечислим события, связанные с объектом `Application`.

Таблица 7.3. *События объекта* `Application`

Событие	Описание
`NewWorkbook`	Создание новой рабочей книги
`SheetActivate`	Активизация листа. Может быть как рабочий лист, так и лист с диаграммой
`SheetBeforeDoubleClick`	Двойной щелчок на рабочем листе
`SheetBeforeRightClick`	Щелчок правой кнопкой мыши на рабочем листе. Событие происходит до того, как генерируется определенное по умолчанию аналогичное событие. Если значение параметра `Cancel` процедуры обработки этого события установить равным `True`, то определенное по умолчанию действие не выполняется
`SheetCalculate`	Пересчет на рабочем листе или изменение данных в диаграмме
`SheetChange`	Изменение данных в ячейках рабочего листа
`SheetDeactivate`	Деактивизация листа. Может быть как рабочий лист, так и лист с диаграммой

Таблица 7.3 *(окончание)*

Событие	Описание
SheetFollowHyperlink	Щелчок на гиперссылке
SheetPivotTableUpdate	Обновление сводной таблицы
SheetSelectionChange	Смена выделения на рабочем листе
WindowActivate	Активизация окна
WindowDeactivate	Деактивизация окна
WindowResize	Изменение размеров окна
WorkbookActivate	Активизация рабочей книги
WorkbookAddinInstall	Инсталляция надстройки
WorkbookAddinUninstall	Удаление надстройки
WorkbookBeforeClose	Перед закрытием рабочей книги. Если значение параметра Cancel процедуры обработки этого события установить равным True, то по завершению ее работы книга не закроется
WorkbookBeforePrint	Перед печатью рабочей книги
WorkbookBeforeSave	Перед сохранением рабочей книги
WorkbookDeactivate	Деактивация рабочей книги
WorkbookNewSheet	Добавление нового листа в рабочую книгу
WorkbookOpen	Открытие рабочей книги
WorkbookPivotTableCloseConnection	Закрытие связей со сводной таблицей
WorkbookPivotTableOpenConnection	Установка связей со сводной таблицей

Отключение генерации событий

Свойство EnableEvents объекта Application позволяет отключать генерацию событий. Например, следующие три инструкции обеспечивают сохранение рабочей книги без отображения предупреждающего сообщения.

```
Application.EnableEvents = False
ActiveWorkbook.Save
Application.EnableEvents = True
```

Семейство *Workbooks*

Семейство `Workbooks` инкапсулирует в себе все рабочие листы рабочей книги. В этом семействе имеются два основных свойства: `Count` возвращает число элементов семейства, а `Item` — конкретный элемент семейства. Методы семейства `Workbooks` позволяют открывать существующую книгу (метод `Open`), закрывать ее (метод `Close`), создавать новую книгу (метод `Add`), причем как пустую, так и заполненную на основе данных либо из текстового файла (метод `OpenText`), либо из xml-файла (метод `OpenXML`), либо из базы данных (метод `OpenDatabase`).

Создание новой рабочей книги

Метод `Add` семейства `Workbooks` создает новую рабочую книгу.

`Add(`*`Template`*`)`

Здесь *`Template`* — необязательный параметр, определяющий то, какой будет создана новая книга. В качестве значения это параметра можно указать:

- имя файла, который будет использован в качества шаблона для создаваемой книги;
- одну из следующих констант `XlWBATemplate`: `xlWBATChart`, `xlWBATExcel4IntlMacroSheet`, `xlWBATExcel4MacroSheet` и `xlWBATWorksheet`.

Если этот параметр опущен, то создается рабочая книга с числом листов, заданных свойством `SheetsInNewWorkbook`.

Открытие рабочей книги

Метод `Open` семейства `Workbooks` открывает указанную рабочую книгу.

`Open(`*`FileName`*`, `*`UpdateLinks`*`, `*`ReadOnly`*`, `*`Format`*`, `*`Password`*`, `*`WriteResPassword`*`, `*`IgnoreReadOnlyRecommended`*`, `*`Origin`*`, `*`Delimiter`*`, `*`Editable`*`, `*`Notify`*`, `*`Converter`*`, `*`AddToMru`*`, `*`Local`*`, `*`CorruptLoad`*`, `*`OpenConflictDocument`*`)`, где:

- *`FileName`* — обязательный параметр, специфицирующий имя файла;
- *`UpdateLinks`* — необязательный параметр, указывающий на то, как должны обновляться связи в рабочей книге;
- `ReadOnly` — необязательный параметр. Если он принимает значение `True`, то книга открывается только для чтения;
- *`Format`* — необязательный параметр, специфицирующий разделитель, если MS Excel открывает текстовый файл;
- *`Password`* — необязательный параметр, указывающий пароль для открытия защищенной рабочей книги;
- *`WriteResPassword`* — необязательный параметр, специфицирующий пароль для внесения изменений в защищенную от записи книгу;

- *IgnoreReadOnlyRecommended* — необязательный параметр, указывающий, надо ли отображать предупреждение о том, что книга была сохранена в режиме "только для чтения";
- *Origin* — необязательный параметр, специфицирующий, какой текстовый файл создан, если MS Excel открывает текстовый файл;
- *Delimiter* — необязательный параметр, указывающий пользовательский разделитель. Этот параметр используется только в случае, если значение параметра *Format* равно 6;
- *Editable* — необязательный параметр, позволяющий открыть данный шаблон для редактирования;
- *Notify* — необязательный параметр, который при значении False запрещает любые попытки открытия недостижимых по каким-либо причинам файлов;
- *Converter* — необязательный параметр, специфицирующий конвертор, если таковой требуется при открытии файла;
- *AddToMru* — необязательный параметр, значение True которого обеспечивает добавление файла в список недавно использованных файлов;
- *Local* — необязательный параметр, специфицирующий, надо ли сохранять файл в соответствии с локальными установками MS Excel;
- *CorruptLoad* — необязательный параметр, указывающий, открывается нормальный или восстановленный файл;
- *OpenConflictDocument* — необязательный параметр, позволяющий открыть конфликтные документы.

Например, следующая инструкция открывает рабочую книгу Report.xls.

```
Workbooks.Open "Report.xls".
```

Объект *Workbook*

В иерархии MS Excel объект Workbook идет сразу после объекта Application и представляет файл рабочей книги. Рабочая книга хранится в файлах формата XLS (стандартная рабочая книга), либо XLA (полностью откомпилированное приложение). Объект Workbook возвращается либо как элемент семейства Workbooks соответствующим свойством, либо как активная рабочая книга — свойством ActiveWorkbook, либо как книга, в которой данный код выполняется — свойством ThisWorkbook.

Свойства объекта *Workbook*

Объект Workbook обладает большим набором свойств. Свойства ActiveChart и ActiveSheet возвращают активную диаграмму и лист. Свойства Charts, Comments, Names, Sheets, Windows, Worksheets возвращают семейства диа-

грамм, примечаний, имен, листов, окон и рабочих листов данной книги. Перечислим основные свойства объекта `Workbook` (табл. 7.4).

Таблица 7.4. *Основные свойства объекта* `Workbook`

Свойство	Описание
`ActiveChart`	Возвращает активную диаграмму
`ActiveSheet`	Возвращает активный лист книги. В следующем примере устанавливается имя активного рабочего листа `ActiveSheet.Name = "Отчет"`
`Author`	Устанавливает имя автора книги
`AutoUpdateFrequency`	Задает частоту обновления книги, совместно используемой несколькими пользователями
`AutoUpdateSaveChanges`	Указывает, надо ли автоматически пересылать обновления другому пользователю при их сохранении и совместном использовании книги
`CalculationVersion`	Возвращает версию MS Excel, в которой рабочая книга в последний раз была полностью пересчитана
`ChangeHistoryDuration`	Задает число дней, в течение которых запоминаются изменения в книге при ее совместном использовании
`Charts`	Возвращает семейство всех диаграмм книги (которые не внедрены в рабочие листы)
`CommandBars`	Возвращает семейство всех панелей управления
`Comments`	Возвращает семейство всех примечаний
`CreateBackup`	Устанавливает, надо ли создавать backup-копию файла при сохранении изменений в книге
`DisplayDrawingObjects`	Управляет отображением графических объектов. Допустимыми значениями являются следующие константы `XlDisplayDrawingObjects`: `xlDisplayShapes`, `xlPlaceholders` и `xlHide`
`EnableAutoRecover`	Позволяет автоматически восстанавливать поврежденную книгу

Таблица 7.4 *(продолжение)*

Свойство	Описание
FileFormat	Задает формат файла рабочей книги
FullName	Задает полное имя рабочей книги
FullNameURLEncoded	Задает имя объекта совместно с указанием пути, где расположен файл
HTMLProject	Возвращает объект HTMLProject, представляющий HTML-проект книги в MS Script Editor
HasPassword	Возвращает значение True, если книга защищена паролем, и значение False в противном случае
HasRoutingSlip	Проверяет, ассоциирована ли с книгой маршрутизация ее рассылки пользователям
HighlightChangesOnScreen	Выделяет на экране изменения при совместной работе с книгой нескольких пользователей
IsAddin	Проверяет, используется ли книга как надстройка
KeepChangeHistory	Проверяет, запоминаются ли исправления, произведенные пользователями при совместной работе над книгой
ListChangesOnNewSheet	Если значение свойства равно True, то исправления, произведенные пользователями при совместной работе над книгой, отображаются на новом рабочем листе
MultiUserEditing	Если значение свойства равно True, то открытая книга разрешена для совместного использования несколькими пользователями
Name	Задает имя книги
Names	Задает семейство всех имен, использованных в книге
Password	Задает пароль для защиты книги
PasswordEncryptionAlgorithm	Задает алгоритм шифрования, используемый MS Excel для кодировки пароля
PasswordEncryptionFileProperties	Проверяет, шифруются ли свойства документа

Таблица 7.4 *(продолжение)*

Свойство	Описание
PasswordEncryptionKeyLength	Задает длину ключа
PasswordEncryptionProvider	Задает тип шифрования, используемого MS Excel для кодировки пароля
Path	Возвращает путь к книге
PrecisionAsDisplayed	Задает, надо ли производить операции с данными с той точностью, как они отображаются на рабочем листе
ProtectStructure	Проверяет, защищена ли структура рабочей книги
ProtectWindows	Проверяет, защищены ли окна рабочей книги
PublishObjects	Возвращает семейство PublishObjects объектов PublishObject, каждый из которых представляет элемент рабочей книги, сохраненный для использования в Web-странице
ReadOnly	Проверяет, открыта ли книга в режиме "только для чтения"
RevisionNumber	Возвращает количество сохранений книги, пока она была открыта для совместного использования
Routed	Проверяет, была ли книга отправлена по маршруту следующему пользователю
RoutingSlip	Возвращает объект RoutingSlip, который инкапсулирует в себе данные, необходимые при маршрутизированной рассылке книги
SaveLinkValues	Сохранение внешних связей вместе с книгой
Saved	Возвращает значение True, если никаких изменений не было сделано в книге с момента ее последнего сохранения, и значение False в противном случае
Sheets	Возвращает семейство всех листов книги
ShowPivotTableFieldList	Проверяет, можно ли отобразить список полей сводной таблицы
SmartTagOptions	Возвращает объект SmartTagOptions, инкапсулирующий в себе свойства смарт-тега

Таблица 7.4 *(окончание)*

Свойство	Описание
`Styles`	Возвращает семейство всех стилей, используемых в книге
`Title`	Задает заголовок Web-страницы, когда книга будет сохранена как Web-страница
`UpdateLinks`	Указывает на то, как обновляются связи с вложенными в книгу OLE-объектами. Допустимы следующие константы `XlUpdateLinks`: `xlUpdateLinksAlways`, `xlUpdateLinksNever`, `xlUpdateLinksUserSetting`
`UserControl`	Возвращает значение `True`, если книга была создана пользователем, и значение `False`, если она была сконструирована программно
`WebOptions`	Возвращает объект `WebOptions`, который инкапсулирует в себе данные, используемые MS Excel при сохранении книги как Web-страницы
`Windows`	Возвращает семейство окон книги
`Worksheets`	Возвращает семейство всех рабочих листов книги
`WritePassword`	Устанавливает пароль для записи книги
`WriteReserved`	Проверяет, защищена ли книга от записи
`WriteReservedBy`	Задает имя пользователя, у которого в данный момент имеются полномочия на защиту книги от записи

Закрытие книги без сохранения изменений

Свойство `Saved` объекта `Workbook` возвращает значение `True`, если в книге не было сделано никаких изменений с момента ее последнего сохранения, и значение `False` в противном случае. Поэтому, если в процедуру обработки события `BeforeClose` объекта `Workbook` добавить инструкцию:

```
ActiveWorkbook.Saved = True,
```

то книга будет сохранена без сохранения изменений, внесенных в нее с момента последнего ее сохранения (листинг 7.27).

Листинг 7.27. Закрытие книги без сохранения изменений. Модуль ЭтаКнига

```
Private Sub Workbook_BeforeClose(Cancel As Boolean)
   ActiveWorkbook.Saved = True
End Sub
```

Определение объекта *Workbook* по имени рабочей книги

Объекты `Workbook` образуют семейство `Workbooks`, доступ к элементам которого можно производить как по имени, так и индексу. В следующем коде доступ к элементам этого семейства реализуется по имени. В качестве искомой книги взята активная рабочая книга, ссылка на которую возвращается свойством `ActiveWorkbook`. Свойство `FullName` возвращает полное имя книги, а метод `SaveAs` сохраняет ее по указанному имени.

```
Dim wb As Workbook
Set wb = Workbooks(Dir(ActiveWorkbook.FullName))
Debug.Print wb.FullName
wb.SaveAs Filename:="newfile.xls"
Debug.Print wb.FullName
```

Объект *Name* и простой способ удаления из рабочей книги ненужных имен

Объект `Name` инкапсулирует в себе информацию об имени, используемом в рабочей книге. Все имена образуют семейство `Names`. Удалять ненужные имена из рабочей книги, конечно, можно вручную, но этот процесс довольно утомительный. Код из листинга 7.28 демонстрирует, как семейство `Names` позволяет пролистать все имена и удалить выбранные, а код из листинга 7.29 — как присвоить имя выделенному диапазону.

Листинг 7.28. Просмотр имен

```
Sub CheckingNames()
   Dim nm As Name
   Dim ans As Integer
   If ThisWorkbook.Names.Count = 0 Then
      MsgBox "Имена не определены."
      Exit Sub
   End If
   For Each nm In ThisWorkbook.Names
      With nm
```

```
        ans = MsgBox("Удалить имя " & .Name & " ? " & vbCr & _
              "относящее к  " & .RefersTo, vbYesNo + vbQuestion)
        Select Case ans
           Case vbYes
              .Delete
           Case vbNo
        End Select
      End With
   Next
End Sub
```

Листинг 7.29. Присвоение имени выделенному диапазону

```
Sub AddName()
   Dim name As String
   name = InputBox("Введите имя")
   ActiveSheet.Names.Add Name:=name, RefersTo:="=" & Selection.Address()
End Sub
```

Методы объекта *Workbook*

Объект `Workbook` имеет обширную коллекцию методов, позволяющих реализовывать большой спектр операций от активизации книги до составления сложного отчета с помощью сводной таблицы. В табл. 7.5 перечислены основные методы объекта `Workbook`.

Таблица 7.5. *Основные методы объекта* `Workbook`

Метод	Описание
`AcceptAllChanges`	Принимает все изменения в совместно используемой книге
`Activate`	Активизирует рабочую книгу так, что ее первый рабочий лист становится активным
`AddToFavorites`	Добавляет книгу в **Избранное**
`BreakLink`	Конвертирует формулы или OLE-объекты между связанными книгами в значения
`CanCheckIn`	Проверяет и устанавливает, можно ли редактировать книгу, расположенную на сервере
`ChangeFileAccess`	Изменяет режим доступа к книге

Таблица 7.5 *(продолжение)*

Метод	Описание
ChangeLink	Изменяет тип связи, установленный между документами
CheckIn	Производит редактирование книги, расположенной на сервере
Close	Закрывает книгу
DeleteNumberFormat	Удаляет пользовательский числовой формат
EndReview	Прекращает редактирование файла, который был отослан методом SendForReview
ExclusiveAccess	Устанавливает для текущего пользователя эксклюзивный доступ к книге
FollowHyperlink	Производит переход по гиперссылке
NewWindow	Открывает новое окно в книге
PivotCaches	Возвращает семейство PivotCaches, представляющее собой кэш-таблицу, отведенную под сводную таблицу
PivotTableWizard	Создает сводную таблицу
Post	Пересылает книгу в папку общего доступа. Метод работает только с клиентами MS Exchange, соединенными с MS Exchange-сервером
PrintOut	Печатает книги
PrintPreview	Задает предварительный просмотр книги
Protect	Защищает рабочую книгу от внесения в нее изменений
ProtectSharing	Защищает книгу при совместном использовании
PurgeChangeHistoryNow	Удаляет данные старше указанного срока из журнала изменений
RefreshAll	Обновляет все внешние данные и отчеты, созданные на основе сводной таблицы
RejectAllChanges	Отменяет все изменения в книге при совместном ее использовании
ReloadAs	Перегружает книгу, созданную на основе HTML-документа, используя указанную кодировку
RemoveUser	Отключает указанного пользователя от книги при ее совместном использовании
ReplyWithChanges	Отсылает сообщения, подтверждающие завершение редактирования книги

Таблица 7.5 *(окончание)*

Метод	Описание
`Save`	Сохраняет книгу
`SaveAs`	Сохраняет книгу в другой файл
`SaveCopyAs`	Сохраняет копию книги
`SendMail`	Отсылает почту, используя встроенные средства MS Mail
`Unprotect`	Снимает защиту с рабочей книги
`UnprotectSharing`	Снимает защиту с совместно используемой книги
`UpdateFromFile`	Обновляет из файла книги только для чтения
`UpdateLink`	Обновляет связи MS Excel и OLE
`WebPagePreview`	Задает предварительный просмотр книги, прежде чем она будет сохранена как Web-страница

Установка и снятие защиты книги

Защита на рабочую книгу от внесения в нее изменений устанавливается методом `Protect`, а снимается методом `Unprotect`.

`Protect(`*`Password`*`,` *`Structure`*`,` *`Windows`*`)`, где:

- *`Password`* — необязательный параметр, задающий пароль для защиты книги. Если параметр опущен, то книга защищена без пароля;
- *`Structure`* — необязательный параметр, устанавливающий, защищена ли структура книги, т. е. взаимное расположение листов;
- *`Windows`* — необязательный параметр, определяющий, защищены ли окна книги.

Защита с рабочей книги снимается методом `Unprotect`.

`Unprotect(`*`Password`*`)`

Здесь *`Password`* — необязательный параметр, указывающий пароль защиты.

В приводимом ниже коде (листинг 7.30) обрабатывается событие `BeforeRightClick` объекта `Worksheet`. При щелчке правой кнопкой на ячейке **A1** устанавливается защита структуры и окон рабочей книги. Щелчок правой же кнопкой на ячейке **B1** снимает эту защиту.

Листинг 7.30. Установка и снятие защиты книги. Модуль рабочего листа

```
Private Sub Worksheet_BeforeRightClick(ByVal Target As Range, _
                                       Cancel As Boolean)
   If Target.Address = "$A$1" Then
```

```
      Cancel = True
      ThisWorkbook.Protect Password:="wow", Structure:=True, _
                           Windows:=True
   End If
   If Target.Address = "$B$1" Then
      Cancel = True
      ThisWorkbook.Unprotect Password:="wow"
   End If
End Sub
```

Как сохранить рабочую книгу, чтобы ее именем была текущая дата

Метод `SaveAs` объекта `Workbook` производит сохранение рабочей книги. Если необходимо, чтобы книга сохранилась так, чтобы ее именем была текущая дата, то можно воспользоваться следующим кодом (листинг 7.31).

Листинг 7.31. Сохранение рабочей книги

```
Sub SaveAsTodaysDate()
   Dim FileName As String, Path As String
   Path = "C:\"
   FileName = Format(Now(), "mmddyy")
   ActiveWorkbook.SaveAs Path & FileName
End Sub
```

Как определить, была ли сохранена открытая рабочая книга

Для того чтобы определить была ли рабочая книга сохранена, можно проверить длину строки, возвращаемую свойством `Path` объекта `Workbook`, как показано в листинге 7.32. Это свойство возвращает полный путь к приложению без последнего разделителя и имени файла. Если это свойство возвращает значение 0, то приложение еще не было сохранено.

Листинг 7.32. Определение, была ли сохранена открытая рабочая книга

```
Function IsSaved() As Boolean
   If Len(ThisWorkbook.Path) = 0 Then
       IsSaved = False
   Else
      IsSaved = True
   End If
End Function
```

События объекта *Workbook*

У объекта `Workbook` имеется большая коллекция событий, перечисленных в табл. 7.6. Они позволяют управлять процессом открытия книги, печатью, отслеживать любые изменения, произведенные в ней, и т. д.

Таблица 7.6. *События объекта* `Workbook`

Событие	Описание
`Activate`	Генерируется при активизации рабочей книги
`AddinInstall`	Генерируется, когда рабочая книга инсталлируется как надстройка
`AddinUninstall`	Генерируется, когда рабочая книга удаляется как инсталлированная надстройка
`BeforeClose`	Генерируется перед закрытием рабочей книги
`BeforePrint`	Генерируется перед печатью
`BeforeSave`	Генерируется перед сохранением рабочей книги
`Deactivate`	Генерируется при потере фокуса рабочей книгой
`NewSheet`	Генерируется при создании нового листа
`Open`	Генерируется при открытии рабочей книги
`PivotTableCloseConnection`	Генерируется при закрытии связи сводной таблицы с источником данных
`PivotTableOpenConnection`	Генерируется при установке связи сводной таблицы с источником данных
`SheetActivate`	Генерируется при активизации любого рабочего листа
`SheetBeforeDoubleClick`	Генерируется при двойном щелчке, но перед выполнением действий, назначенных по умолчанию двойному щелчку
`SheetBeforeRightClick`	Генерируется при щелчке правой кнопкой мыши, но перед выполнением действий, назначенных по умолчанию этому щелчку
`SheetCalculate`	Генерируется после завершения всех вычислений во всех рабочих листах и перерисовки диаграмм
`SheetChange`	Генерируется при изменении значения на любом листе в какой-либо ячейке
`SheetDeactivate`	Генерируется при потере листом фокуса
`SheetFollowHyperlink`	Генерируется при щелчке на гиперссылке

Таблица 7.6 *(окончание)*

Событие	Описание
SheetPivotTableUpdate	Генерируется после обновления сводной таблицы
SheetSelectionChange	Генерируется при изменении выделения на любом листе
WindowActivate	Генерируется при активизации окна
WindowDeactivate	Генерируется при деактивизации окна
WindowResize	Генерируется при изменении размеров окна

Запрет закрытия рабочей книги

У процедуры обработки события `BeforeClose` объекта `Workbook`, генерируемого перед закрытием рабочей книги, имеется параметр `Cancel`. При значении `True` он блокирует закрытие книги. При значении `False` — закрытие разрешено. Например, следующий код (листинг 7.33) позволяет пользователю закрыть книгу только в случае, если в ячейку **A1** введена строка `"Go"`.

Листинг 7.33. Запрет закрытия рабочей книги. Модуль `ЭтаКнига`

```
Private Sub Workbook_BeforeClose(Cancel As Boolean)
   If Range("A1").Value = "Go" Then
      Cancel = False
   Else
     Cancel = True
   End If
End Sub
```

Семейство *Worksheets*

Семейство `Worksheets` инкапсулирует в себе все листы рабочей книги. Основные свойства этого семейства возвращают число элементов семейства и конкретные экземпляры. Методы же семейства `Worksheets` предоставляют более широкий ассортимент операций, начиная от создания нового листа и заканчивая его удалением. Свойства и методы семейства `Worksheets` перечислены в табл. 7.7 и табл. 7.8.

Таблица 7.7. *Свойства семейства* `Worksheets`

Свойство	Описание
`Count`	Возвращает число элементов семейства
`Item`	Возвращает элемент семейства

Таблица 7.8. *Методы семейства* `Worksheets`

Метод	Описание
`Add`	Создает новый лист. `Add (Before, After, Count, Type)`, где: *`Before`* — необязательный параметр, специфицирующий лист, перед которым будет добавлен новый; *`After`* — необязательный параметр, специфицирующий лист, после которого будет добавлен новый; *`Count`* — необязательный параметр, указывающий число добавляемых листов. По умолчанию его значение равно единице; *`Type`* — необязательный параметр, задающий тип листа. Его допустимым значением может быть одна из следующих констант `XlSheetType`: `xlWorksheet`, `xlChart`, `xlExcel4MacroSheet` и `xlExcel4IntlMacroSheet`. По умолчанию подразумевается значение `xlWorksheet`
`Copy`	Копирует лист. `Copy (`*`Before`*`, `*`After`*`)`, где: *`Before`* — необязательный параметр, специфицирующий лист, перед которым будет вставлен данный; *`After`* — необязательный параметр, специфицирующий лист, после которого будет вставлен данный
`Delete`	Удаляет лист
`Move`	Перемещает лист. `Move (Before, After)` , где: *`Before`* — необязательный параметр, специфицирующий лист, перед которым будет вставлен данный; *`After`* — необязательный параметр, специфицирующий лист, после которого будет вставлен данный

Таблица 7.8 *(окончание)*

Метод	Описание
PrintOut	Печать листа или группы листов. PrintOut (*From*, *To*, *Copies*, *Preview*, ↳*ActivePrinter*, *PrintToFile*, *Collate*, ↳*PrToFileName*), где: *From* — необязательный параметр, определяющий номер листа, с которого начинается печать группы специфицированных рабочих листов; *To* — необязательный параметр, специфицирующий номер последнего листа из группы печатаемых листов; *Copies* — необязательный параметр, задающий число печатаемых копий; *Preview* — необязательный параметр, причем, если его значение равно True, то перед осуществлением печати MS Excel отображает окно предварительного просмотра, а если False — то печать происходит непосредственно без отображения этого окна; *ActivePrinter* — необязательный параметр, задающий имя активного принтера; *PrintToFile* — необязательный параметр, причем, если его значение равно True, то печать производится в файл; *Collate* — необязательный параметр, причем, если его значение равно True, MS Excel разбирает документ по печатаемым копиям; *PrToFileName* — необязательный параметр, указывающий имя выходного файла, если значение параметра *PrintToFile* установлено равным True
PrintPreview	Отображение окна предварительного просмотра
Select	Выбирает специфицированный лист
FillAcrossSheets	Копирует указанный диапазон в другие рабочие листы данной коллекции. FillAcrossSheets (*Range*, *Type*), где: *Range* — обязательный параметр, дающий ссылку на копируемый диапазон; *Type* — необязательный параметр, специфицирующий то, как диапазон будет копироваться. Допустимые значения определяются константами XlFillWith: xlFillWithAll, xlFillWithContents и xlFillWithFormats

Вставка нового листа с именем, отличным от существующих

В качестве примера приведем текст процедуры (листинг 7.34), которая добавляет в книгу новый рабочий лист, причем первоначально проверяет, существует ли уже лист со специфицированным именем. Если такой лист имеется или пользователь забыл определить имя добавляемого листа, то отображается сообщение, и процедура завершает работу.

Листинг 7.34. Вставка нового листа с именем, отличным от существующих

```
Sub AddNewSheet(newSheetName As String)
   Dim ws As Worksheet
   For Each ws In Worksheets
      If ws.name = newSheetName Or newSheetName = "" Then
        MsgBox "Такой лист уже существует", vbInformation
        Exit Sub
      End If
   Next
   Sheets.Add Type:="Worksheet"
   With ActiveSheet
      .Move After:=Worksheets(Worksheets.Count)
      .Name = newSheetName
   End With
End Sub
```

Объект *Worksheet*

В иерархии MS Excel объект `Worksheet` идет сразу после объекта `Workbook` и представляет собой рабочий лист. Объект `Worksheet` возвращается либо как элемент семейства `Worksheets`, либо как элемент семейства `Sheets` одноименным свойством объекта `Workbook`, либо как активный рабочий лист — свойством `ActiveSheet`.

Свойства объекта *Worksheet*

Объект `Worksheet` обладает огромной коллекцией свойств, предоставляющих разработчику обширные средства от получения доступа к конкретной ячейке до сложной обработки данных типа "фильтрация". В табл. 7.9 перечислены свойства этого объекта.

Таблица 7.9. *Свойства объекта* `Worksheet`

Свойство	Описание
`AutoFilter`	Возвращает объект `AutoFilter`, инкапсулирующий в себе данные об используемом автофильтре для обработки данных, или значение `Nothing`, если фильтр не применяется
`AutoFilterMode`	Возвращает значение `True`, если в ячейках отображаются раскрывающиеся списки автофильтра
`Cells`	Возвращает одну ячейку, либо семейство ячеек
`CircularReference`	Возвращает указатель на диапазон, содержащий циклическую ссылку
`Columns`	Возвращает либо один столбец, либо семейство столбцов
`Comments`	Возвращает семейство `Comments` всех примечаний
`ConsolidationFunction`	Возвращает код функции консолидации данных, используемой при построении итоговой таблицы
`ConsolidationOptions`	Возвращает трехмерный массив, содержащий в себе данные о конструируемой итоговой таблице
`ConsolidationSources`	Возвращает трехмерный массив, содержащий в себе информацию о расположении консолидируемых данных
`DisplayPageBreaks`	Управляет отображением линий разрыва страницы
`EnableAutoFilter`	Если значение этого свойства равно `True`, то раскрывающиеся списки автофильтра доступны для пользователя даже в случае установки защиты на рабочем листе. При этом должен быть только включен режим защиты пользовательского интерфейса
`EnableCalculation`	Если значение этого свойства равно `False`, то не происходит пересчет формул на рабочем листе, если же его значение равно `True`, то пересчет производится автоматически
`EnableOutlining`	Если значение этого свойства равно `True`, то управление структурой достижимо для пользователя даже в случае установки защиты на рабочем листе. При этом должен быть только включен режим защиты пользовательского интерфейса
`EnablePivotTable`	Если значение этого свойства равно `True`, то управление сводной таблицей достижимо для пользователя даже в случае установки защиты на рабочем листе. При этом должен быть только включен режим защиты пользовательского интерфейса

Таблица 7.9 *(продолжение)*

Свойство	Описание
EnableSelection	Определяет, какие ячейки могут быть выбраны на рабочем листе, если на нем установлена защита. Допустимыми значениями являются следующие константы XlEnableSelection: xlNoSelection (выбор запрещен), xlNoRestrictions (выбирать можно все ячейки), xlUnlockedCells (выбирать можно только те ячейки, у которых значение свойства Locked установлено равным False)
FilterMode	Это свойство принимает значение True, если на рабочем листе имеются списки, к которым были применены фильтры, в результате чего некоторые их элементы оказались скрытыми
HPageBreaks	Возвращает семейство HpageBreaks, состоящее из объектов HPageBreak, инкапсулирующих в себе информацию о горизонтальных разрывах страницы
Hyperlinks	Возвращает семейство Hyperlinks, состоящее из объектов Hyperlink, инкапсулирующих в себе информацию о гиперссылках
Index	Возвращает значения индекса рабочего листа в семействе Sheets
MailEnvelope	Заголовок отсылаемого по электронной почте сообщения, созданного на основе рабочего листа
Name	Имя рабочего листа
Names	Возвращает семейство Names всех имен данного рабочего листа
Outline	Возвращает объект Outline, инкапсулирующий в себе информацию о структуре, созданной на рабочем листе
PageSetup	Возвращает объект PageSetup, инкапсулирующий в себе информацию о параметрах страницы
ProtectContents	Свойство только для чтения, проверяющее установлена ли защита на содержимое рабочего листа
ProtectDrawingObjects	Свойство только для чтения, проверяющее установлена ли защита на графические объекты
ProtectScenarios	Свойство только для чтения, проверяющее установлена ли защита на сценарии
Protection	Возвращает объект Protection, инкапсулирующий в себе информацию об установленной на листе защите

Таблица 7.9 *(окончание)*

Свойство	Описание
ProtectionMode	Свойство только для чтения, проверяющее установлена ли защита только пользовательского интерфейса
QueryTables	Возвращает семейство QueryTables, состоящее из объектов QueryTable, инкапсулирующих в себе информацию о таблицах, построенных на основе запроса к внешним базам данных
Range	Возвращает объект Range, представляющий диапазон
Rows	Возвращает либо одну строку, либо семейство строк
Scripts	Возвращает семейство Scripts, состоящее из объектов Script, инкапсулирующих в себе информацию об используемых скриптах, когда лист сохраняется как Web-страница
ScrollArea	Задает диапазон, в котором разрешена прокрутка
Shapes	Возвращает семейство Shapes, состоящее из объектов Shape, инкапсулирующих в себе информацию о расположенных на листе фигурах
SmartTags	Возвращает семейство SmartTags, состоящее из объектов SmartTag, инкапсулирующих в себе информацию о смарт-тегах
StandardHeight	Возвращает стандартную высоту ячеек
StandardWidth	Возвращает стандартную ширину ячеек
Tab	Возвращает объект Tab, инкапсулирующий в себе информацию о теге листа
Type	Определяет тип листа. Допустимыми значениями могут быть следующие константы XlSheetType: xlChart, xlDialogSheet, xlExcel4IntlMacroSheet, xlExcel4MacroSheet и xlWorksheet
UsedRange	Возвращает объект Range, который содержит данные
VPageBreaks	Возвращает семейство VPageBreaks, состоящее из объектов VPageBreak, инкапсулирующих в себе информацию о вертикальных разрывах страницы
Visible	Управляет видимостью листа. Допустимые значения: True (рабочий лист видим на экране), False (не видим, т. е. скрыт, но его можно отобразить на экране командой **Формат**, **Лист**, **Отобразить**), xlVeryHidden (скрыт, и его можно отобразить на экране только программно)

Проверка, установлена ли защита на содержании рабочего листа

Свойство `ProtectContents` возвращает значение `True`, если содержание рабочего листа защищено. Следующий код (листинг 7.35) проверяет наличие защиты содержания и в случае присутствия таковой, предлагает ее снять.

Листинг 7.35. Проверка, установлена ли защита на содержании рабочего листа

```
Sub DemoProtectContents()
   On Error Resume Next
   If Worksheets(1).ProtectContents Then
      MsgBox "Содержание рабочего листа защищено"
      Worksheets(1).Unprotect
   Else
      MsgBox "Содержание рабочего листа не защищено"
   End If
End Sub
```

А как проверить существует ли лист

Все листы рабочей книги образуют семейство `Sheets`. Для того чтобы определить, существует ли специфицируемый лист, достаточно попытаться получить на него ссылку и, в случае отсутствия таковой, перехватить сгенерируемую ошибку. Следующий код демонстрирует подобный подход (листинг 7.36). Функция `SheetExists` возвращает значение `True`, если данный лист существует, и `False` в противном случае.

Листинг 7.36. Проверка существования листа

```
Function SheetExists(SheetName As String) As Boolean
   Dim obj As Object
   On Error GoTo errorHandler:
   Set obj = Sheets(SheetName)
   SheetExists = True
   Exit Function
errorHandler:
   SheetExists = False
End Function
```

Методы объекта *Worksheet*

Объект `Worksheet` обладает большой коллекцией методов, позволяющих производить широкий спектр действий над данными, начиная от выбора, перемещения и удаления, заканчивая проверкой правописания. В табл. 7.10 перечислим основные методы объекта `Worksheet`.

Таблица 7.10. *Методы объекта* `Worksheet`

Метод	Описание
`Activate`	Активизирует указанный рабочий лист
`Calculate`	Заново вычисляет данные на рабочем листе
`ChartObjects`	Возвращает семейство `ChartObjects` всех вложенных в рабочий лист диаграмм
`CheckSpelling`	Проверка правописания в ячейках, примечаниях, заголовке и объектах, расположенных на рабочем листе
`CircleInvalid`	Обрамление некорректно введенных данных
`ClearArrows`	Удаление стрелок, демонстрирующих зависимости на рабочем листе
`ClearCircles`	Удаление обрамления с некорректно введенных данных
`Copy`	Копирует лист в другое местоположение той же книги
`Delete`	Удаляет лист
`Move`	Перемещает лист
`OLEObjects`	Возвращает семейство `OLEObjects` всех внедренных OLE-объектов
`Paste`	Вставляет из буфера обмена на рабочий лист
`PasteSpecial`	Вставляет данные на рабочий лист, используя специальную вставку
`PivotTableWizard`	Создает сводную таблицу
`PrintOut`	Печатает содержимое рабочего листа
`PrintPreview`	Предварительный просмотр содержимого рабочего листа перед печатью
`Protect`	Устанавливает защиту на рабочий лист
`ResetAllPageBreaks`	Переустанавливает на рабочем листе все разрывы страницы
`SaveAs`	Сохраняет измененную страницу в отдельный файл
`Scenarios`	Возвращает либо объект `Scenario`, представляющий единственный сценарий, либо семейство `Scenarios` всех сценариев

Таблица 7.10 *(окончание)*

Метод	Описание
`Select`	Выбирает указанный рабочий лист
`SetBackgroundPicture`	Устанавливает фоновый рисунок для рабочего листа
`ShowAllData`	Отображает все скрытые при помощи примененных фильтров данные
`Unprotect`	Снимает защиту с рабочего листа

Удаление рабочего листа без предупреждения пользователя

Для удаления рабочего листа без предупреждения пользователя надо временно при помощи свойства `DisplayAlerts` отключить отображение предупреждающего окна, затем удалить требуемый лист. После этого режим отображения предупреждений разумно снова включить. В следующем коде, например, по данной схеме удаляется рабочий лист **Отчет**.

```
Application.DisplayAlerts = False
Worksheets("Отчет").Delete
Application.DisplayAlerts = True
```

Установка и снятие защиты с рабочего листа

Защита на рабочий лист устанавливается методом `Protect`, у которого имеется обширная коллекция необязательных параметров.

`Protect(`*`Password`*`, `*`DrawingObjects`*`, `*`Contents`*`, `*`Scenarios`*`, `*`UserInterfaceOnly`*`, ↳`*`AllowFormattingCells`*`, `*`Allow0FormattingColumns`*`, `*`AllowFormattingRows`*`, ↳`*`AllowInsertingColumns`*`, AllowInsertingRows, `*`AllowInsertingHyperlinks`*`, ↳`*`AllowDeletingColumns`*`, `*`AllowDeletingRows`*`, `*`AllowSorting`*`, `*`AllowFiltering`*`, ↳`*`AllowUsingPivotTables`*`)`, где:

- *`Password`* — задает пароль защиты;
- *`DrawingObjects`* — устанавливает защиту на графические объекты;
- *`Contents`* — определяет защиту на содержание ячеек;
- *`Scenarios`* — задает защиту на сценарии;
- *`UserInterfaceOnly`* — защита пользовательского интерфейса, но не макросов. Если этот параметр опущен, то защита применяется как к интерфейсу, так и к макросам;
- *`AllowFormattingCells`* — разрешение форматировать ячейки;
- *`AllowFormattingColumns`* — разрешение форматировать столбцы;
- *`AllowFormattingRows`* — разрешение форматировать строки;

- *AllowInsertingColumns* — разрешение вставлять столбцы;
- *AllowInsertingRows* — разрешение вставлять строки;
- *AllowInsertingHyperlinks* — разрешение вставлять гиперссылки;
- *AllowDeletingColumns* — разрешение удалять столбцы;
- *AllowDeletingRows* — разрешение удалять строки;
- *AllowSorting* — разрешение сортировать данные;
- *AllowFiltering* — разрешение фильтровать данные;
- *AllowUsingPivotTables* — разрешение конструировать сводные таблицы.

Снятие защиты осуществляется методом

```
Unprotect(Password)
```

Здесь *Password* — пароль защиты.

Объект *Protection* (как определить, какая защита установлена на рабочем листе)

Объект `Protection` инкапсулирует в себе информацию о том, какая защита включена на рабочем листе. Этот объект возвращается свойством `Protection` объекта `Worksheet`. Его свойства, перечисленные в табл. 7.11, как раз и идентифицируют установленную защиту.

Таблица 7.11. *Свойства объекта* `Protection`

Свойство	Описание
AllowDeletingColumns	Разрешено удаление столбцов
AllowDeletingRows	Разрешено удаление строк
AllowFiltering	Разрешена фильтрация
AllowFormattingCells	Разрешено форматирование ячеек
AllowFormattingColumns	Разрешено форматирование столбцов
AllowFormattingRows	Разрешено форматирование строк
AllowInsertingColumns	Разрешена вставка столбцов
AllowInsertingHyperlinks	Разрешена вставка гиперссылок
AllowInsertingRows	Разрешена вставка строк
AllowSorting	Разрешена сортировка
AllowUsingPivotTables	Разрешено использование сводных таблиц

Кроме объекта `Protection` имеется еще ряд свойств рабочего листа, определяющих тип защиты, не покрытых свойствами объекта `Protection` (табл. 7.12).

***Таблица 7.12.** Свойства, проверяющие установленную защиту*

Свойство	Описание
`ProtectContents`	Свойство "только для чтения", проверяющее, установлена ли защита на содержимое рабочего листа
`ProtectDrawingObjects`	Свойство "только для чтения", проверяющее, установлена ли защита на графические объекты
`ProtectScenarios`	Свойство "только для чтения", проверяющее, установлена ли защита на сценарии
`ProtectionMode`	Свойство "только для чтения", проверяющее, установлена ли защита только пользовательского интерфейса

Объект *PageSetup,* и как вывести в колонтитул имя книги и рабочего листа и текущую дату

Объект `PageSetup`, возвращаемый свойством `PageSetup` объекта `Worksheet` позволяет контролировать установки документа перед его печатью. В частности, с помощью него можно установить и колонтитулы страницы. Здесь пригодятся свойства `LeftHeader`, `RightHeader` и `CenterHeader`. Следующий код (листинг 7.37) демонстрирует то, как можно вывести в колонтитул имя книги и рабочего листа и текущую дату.

Листинг 7.37. Вывод данных в колонтитул. Модуль `ЭтаКнига`

```
Sub FormatHeader()
   Dim i As Integer
   With ThisWorkbook
      For i = 1 To .Worksheets.Count - 1
         .Worksheets(i).PageSetup.LeftHeader = .FullName
         .Worksheets(i).PageSetup.RightHeader = Now()
         .Worksheets(i).PageSetup.CenterHeader = Worksheets(i).Name
      Next
   End With
End Sub

Private Sub Workbook_Open()
   FormatHeader
End Sub
```

События объекта *Worksheet*

Объект Worksheet поддерживает множество событий, позволяющих отслеживать всевозможные действия пользователя от активизации листа до перехода по гиперссылке. В табл. 7.13 перечислены события, связанные с объектом `Worksheet`.

Таблица 7.13. *События объекта* `Worksheet`

Событие	Описание
`Activate`	Генерируется при активизации листа
`BeforeDoubleClick`	Генерируется перед двойным щелчком до того, как будет выполнено действие, ассоциированное по умолчанию с соответствующим объектом
`BeforeRightClick`	Генерируется перед щелчком правой кнопкой мыши до того, как будет выполнено действие, ассоциированное по умолчанию с соответствующим объектом
`Calculate`	Генерируется при пересчете данных
`Change`	Генерируется при изменениях данных
`Deactivate`	Генерируется при деактивизации листа
`FollowHyperlink`	Генерируется при переходе по гиперссылке
`PivotTableUpdate`	Генерируется при обновлении сводной таблицы
`SelectionChange`	Генерируется при смене выделения

Блокировка действий, связанных с событием по умолчанию

С теми или иными действиями пользователя связаны по умолчанию некоторые системные действия. Например, при нажатии правой кнопки мыши на рабочем листе отображается стандартное контекстное меню. Естественно, возникает вопрос, как блокировать эти действия, если они не совпадают с бизнес-логикой вашего проекта. Да, очень просто. В процедурах обработки событий часто присутствует параметр `Cancel`, который, если принимает значение `True`, как раз и блокирует эти действия по умолчанию, предлагаемые системой пользователю. Например, в следующем коде (листинг 7.38) блокируется отображение контекстного меню, появляющегося при нажатии правой кнопки мыши. Вместо него в месте на рабочем листе, где был произведен щелчок, добавляется надпись с порядковым номером этого щелчка.

Листинг 7.38. Блокировка отображения контекстного меню, появляющегося при нажатии правой кнопки мыши

```
Private Sub Worksheet_BeforeRightClick(ByVal Target As Range, _
                                       Cancel As Boolean)
   Cancel = True
   Static i As Integer
   Dim x As Integer
   Dim y As Integer
   x = Target.Left
   y = Target.Top
   ActiveSheet.Shapes.AddTextbox(msoTextOrientationHorizontal, _
               x, y, 30, 20).TextFrame.Characters.Text = i
   i = i + 1
End Sub
```

Еще одним примером использования объекта `Shape` может быть следующий код (листинг 7.39), последовательно с интервалом в одну секунду выводящий различные автофигуры, а затем с такой же скоростью их удаляющий (рис. 7.3).

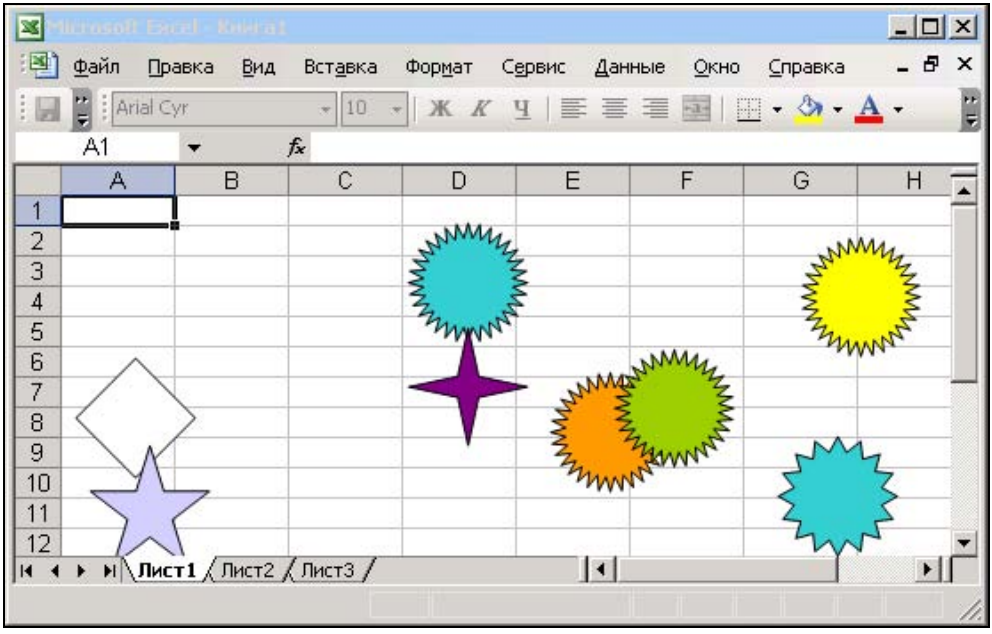


Рис. 7.3. Последовательный вывод автофигур

Листинг 7.39. Последовательный вывод автофигур

```
Sub StarShow()
   Dim w As Integer, h As Integer, i As Integer
   Dim toppos As Integer, leftpos As Integer
   Dim v As Long
   Dim star As Shape
   w = 50: h = 50
   Randomize
   For i = 1 To 10
      toppos = Rnd() * (ActiveWindow.UsableHeight - h)
      leftpos = Rnd() * (ActiveWindow.UsableWidth - w)
      Select Case (i Mod 6)
         Case 0
            v = msoShape4pointStar
         Case 1
            v = msoShape5pointStar
         Case 2
            v = msoShape16pointStar
         Case 3
           v = msoShape32pointStar
         Case 5
           v = msoShapeDiamond
      End Select
      Set star = ActiveSheet.Shapes.AddShape(v, leftpos, toppos, w, h)
      star.Fill.ForeColor.SchemeColor = Int(Rnd() * 56)
      Application.Wait Now + TimeValue("00:00:01")
      DoEvents
   Next
   Application.Wait Now + TimeValue("00:00:01")
   For Each star In Worksheets(1).Shapes
      If Left(star.Name, 9) = "AutoShape" Then
         star.Delete
         DoEvents
         Application.Wait Now + TimeValue("00:00:01")
      End If
   Next
End Sub
```

Автоматическое переоформление таблицы при изменении в ней значений

Событие `Change` объекта `Worksheet` генерируется при изменении значений в диапазоне рабочего листа. Это событие позволяет автоматизировать переоформление таблицы при изменении в ней значений. Рассмотрим следующую бизнес-ситуацию. В диапазоне **B2:B11** расположены данные по расходам за отчетный период фирмы "Родные просторы". Необходимо выделить полужирным красным шрифтом те данные, которые соответствуют максимальному объему расходов, синим же цветом — минимальному. Все остальные данные выводятся черным шрифтом. Кроме того, ячейки, объем расходов в которых превышает средний объем, надо залить желтым цветом. Причем требуется обеспечить автоматическое переформатирование таблицы при изменении значений в ее ячейках (рис. 7.4). Для решения этой задачи как раз и пригодится событие `Change` объекта `Worksheet`, а как это делается, показано в следующем коде (листинг 7.40).

	A	B
1		***Расходы***
2	***Январь***	4565
3	***Февраль***	5676
4	***Март***	4556
5	***Апрель***	4567
6	***Май***	6576
7	***Июнь***	5671
8	***Июль***	**9876**
9	***Август***	5675
10	***Сентябрь***	5676
11	***Октябрь***	7546

Рис. 7.4. Автоматическое переоформление таблицы при изменении в ней значений

Листинг 7.40. Автоматическое переоформление таблицы при изменении в ней значений

```
Private Sub Worksheet_Change(ByVal Target As Range)
   Dim rng As Range
   Dim c As Range
   Dim max As Double
   Dim min As Double
   Dim avr As Double
   Set rng = Range("B2:B11")
   If Not (Application.Intersect(Target, rng) Is Nothing) Then
```

```
      If Application.WorksheetFunction.CountA(rng) > 0 Then
         max = Application.WorksheetFunction.max(rng)
         min = Application.WorksheetFunction.min(rng)
         avr = Application.WorksheetFunction.Average(rng)
         For Each c In rng
            If c.Value = max Then
               c.Font.Bold = True
               c.Font.Color = RGB(255, 0, 0)
            ElseIf c.Value = min Then
               c.Font.Bold = False
               c.Font.Color = RGB(0, 0, 255)
            Else
               c.Font.Bold = False
               c.Font.Color = RGB(0, 0, 0)
            End If
            If c.Value > avr Then
               c.Interior.Color = RGB(255, 255, 0)
            Else
               c.Interior.ColorIndex = xlNone
            End If
         Next
      Else
        rng.Interior.ColorIndex = xlNone
      End If
   End If
End Sub
```

Автоматический ввод данных в верхнем регистре данного диапазона

При помощи обработки события `Change` объекта `Worksheet` можно не только переформатировать данные, но и блокировать действия пользователя. Например, следующий код (листинг 7.41) обеспечит то, что в первом столбце рабочего листа данные всегда будут вводиться в верхнем регистре.

Листинг 7.41. Ввод данных только в верхнем регистре

```
Private Sub Worksheet_Change(ByVal Target As Range)
   Application.EnableEvents = False
   If Not Application.Intersect(Target, Range("A:A")) Is Nothing Then
```

```
      Target.Value = UCase(Target.Value)
   End If
   Application.EnableEvents = True
End Sub
```

Вывод суммы значений из выделенного диапазона в строку состояния

Обрабатывая событие `SelectionChange` объекта `Worksheet` можно сделать так, чтобы в строку состояния выводилась, например, сумма значений из выделенного диапазона. Для этого надо воспользоваться свойством `StatusBar` объекта `Application`, которое задает текст, отображаемый в строке состояния, как показано в листинге 7.42. Если значение этого свойства устанавливается равным `False`, то в строке состояния отображается текст, заданный по умолчанию.

Листинг 7.42. Вывод суммы значений в строку состояния

```
Private Sub Worksheet_SelectionChange(ByVal Target As Range)
   Dim s As Double
   s = Application.WorksheetFunction.Sum(Target)
   If s <> 0 Then
      Application.StatusBar = FormatNumber(s, 3)
   Else
      Application.StatusBar = False
   End If
End Sub
```

Объекты *Range* и *Selection*

В иерархии MS Excel объект `Range` (*диапазон*) идет сразу после объекта `Worksheet`. Объект `Range` является одним из ключевых объектов VBA. Объект `Selection` возникает в VBA двояко — либо как результат работы метода `Select`, либо при вызове свойства `Selection`. Тип получаемого объекта зависит от типа выделенного объекта. Чаще всего объект `Selection` принадлежит классу `Range`, и при работе с ним можно использовать свойства и методы объекта `Range`. Объект `Range` возвращается либо как элемент семейств `Range` или `Cells`, либо свойствами `Range`, `Cells` и `Offset`, либо методами `ActiveCell`, `Intersect` и `Union`.

Адресация ячеек

При работе с объектом `Range` необходимо помнить, как в MS Excel ссылаются на ячейку рабочего листа. Имеются два способа ссылки на ячейки рабочего листа:

- относительная адресация (табл. 7.14), т. е. когда начало координат, задающее нумерацию строк и столбцов, связывается с объектом, вызвавшим `Range`;
- абсолютная адресация (табл. 7.15).

Таблица 7.14. *Относительная адресация*

Формат	Описание
`A1`	Имя ячейки состоит из имени столбца (их 256 — A, B, ..., Z, AB, ..., HZ, IA, ..., IV) и номера (1, ..., 16384). Например, **A1** или **C2**
`R1C1`	Адресация задается индексом строки и индексом столбца. Например, **R1C1** или **R2C3**

Таблица 7.15. *Абсолютная адресация*

Формат	Описание
`A1`	Признаком абсолютной адресации является знак "$", предшествующий имени строки (абсолютная адресация на строку) или столбца (абсолютная адресация на столбец). Например, `$A10`, `A$10` и `$A$10` задают абсолютную адресацию на столбец **A**, строку **10** и ячейку **A10** соответственно
`R1C1`	Указывается смещение по отношению к активной ячейке. Смещение приводится в квадратных скобках, причем знак указывает на направление смещения. Например, если активной ячейкой является **R2C3**, то `R[1]C[-1]` дает ссылку на ячейку **R3C2**

Адрес ячейки рабочего листа является лишь частью полного адреса ячейки, который в общем случае включает имя рабочего листа и адрес книги. При задании полного адреса за именем листа следует знак "!", а адрес книги заключается в скобки. Например,

```
A1
Лист2!A1
[МояКнига.XLS]Лист2!A1
```

В первой строке данного примера дана относительная ссылка на ячейку **A1** активного рабочего листа, во второй — на ячейку **A1** рабочего листа **Лист2** активной книги, а в третьей — на ячейку **A1** рабочего листа **Лист2** книги МояКнига.xls текущего рабочего каталога.

Задание групп строк и столбцов

Если в диапазоне указываются только имена столбцов или строк, то объект `Range` задает диапазон, состоящий из указанных столбцов или строк. Например, `Range("A:C")` задает диапазон, состоящий из столбцов **A**, **B** и **C**, а `Range("2:2")` — из второй строки. Другим способом работы со строками и столбцами являются свойства рабочего листа `Rows` и `Columns`, возвращающие семейства строк и столбцов. Например, столбцом **A** является `Columns(1)`, а второй строкой `Rows(2)`.

Связь объекта *Range* и свойства *Cells* объекта *Worksheet*

Ячейка — это частный случай диапазона, который состоит из единственного элемента. Поэтому, естественно, что объект `Range` позволяет работать как с диапазоном ячеек, так и с одной ячейкой.

Альтернативным способом работы с ячейкой является свойство `Cells` объекта `Worksheet`. Например, ячейку **A2** как объект можно описать двумя равносильными способами: `Range("A2")` и `Cells(1, 2)`.

В свою очередь ячейка, возвращаемая свойством `Cells`, используемым как параметр объекта `Range`, позволяет записывать диапазон в альтернативном виде, который иногда удобен для работы. В качестве примера этой формы записи диапазона приведем следующие две инструкции, возвращающие один и тот же диапазон:

```
Range("A2:C3")
Range(Cells(1,2), Cells(3,3))
```

Примечание

Диапазон, так же как и рабочий лист, обладает свойством `Cells`, которое, если используется без параметров, возвращает множество всех ячеек, входящих в диапазон. Если же оно используется с параметрами, то возвращает конкретную ячейку из диапазона. Например, в следующем примере значение 2 вводится в ячейку **C3**:

```
Range("B2:D4").Select
Selection.Cells(2, 2).Value = 2
```

При работе с объектом `Range` допустимо использование именных диапазонов. Предположим, что определен диапазон с именем `Отчет`, тогда к нему можно ссылаться следующим образом:

```
Range("Отчет")
```

Если имеются две ячейки, именованные FirstCell и LastCell, то минимальный прямоугольный диапазон, содержащий эти ячейки, может быть описан следующим образом:

```
Range(Range("FirstCell"), Range("LastCell"))
```

При работе с диапазонами допустима ссылка на них по их имени, окруженному квадратными скобками, что является сокращенной записью метода `Evaluate` объекта `Application`.

```
[B1]
[A1:C5]
[A1:A10,C1:C10]
[Продажи]
```

Ссылка на диапазон неактивного рабочего листа производится следующим образом:

```
Worksheets("Май").Range("Отчет")
Worksheets("Май").Range("A1")
```

Свойства объекта *Range*

Объект `Range` позволяет сочетать гибкость VBA и мощь рабочего листа. Огромное число встроенных функций рабочего листа существенно упрощает и делает более наглядным программирование на VBA. Свойства объекта `Range` позволяют управлять им от внешнего вида до автоматизации вычислений. В последующих подразделах на содержательных примерах досконально разобраны следующие свойства этого объекта.

Address	AllowEdit	Areas
Borders	Cells	Characters
Column	Columns	ColumnWidth
Comment	Count	CurrentRegion
End	EntireColumn	EntireRow
Font	Formula	FormulaArray
FormulaHidden	FormulaLocal	FormulaR1C1
FormulaR1C1Local	HasFormula	Height
Hidden	HorizontalAlignment	Hyperlinks
Interior	Left	Locked
Name	NumberFormat	Offset
Orientation	Resize	Row
RowHeight	Rows	ShrinkToFit
Top	UseStandardHeight	UseStandardWidth
Value	VerticalAlignment	Width
Worksheet	WrapText	

Ввод или считывание значения из диапазона

Свойство `Value` объекта `Range` возвращает или устанавливает значение в ячейках диапазона. В первой инструкции данного примера переменной `x`

присваивается значение из ячейки **C1**, во второй в ячейку **C3** вводится строка "Отчет", а в третьей в каждую из ячеек диапазона **A1:B2** вводится 1.

```
x = Range("C1").Value
Range("C3").Value = "Отчет"
Range("A1:B2").Value = 1
```

Поиск по шаблону подобных значений в диапазоне

Последовательный перебор ячеек диапазона и сравнение с помощью оператора `Like` возвращаемых значений свойства `Value` с шаблоном позволяет реализовывать поиск подобных значений в диапазоне. Например, в следующем коде (листинг 7.43) последовательно просматриваются все ячейки диапазона **A1:A100**. В тех из них, в которые входит значение `MS`, содержимое ячейки заменяется словом `Microsoft`, сама же ячейка заливается желтым цветом, в то время как все остальные ячейки — белым цветом.

Листинг 7.43. Поиск в диапазоне подобных значений

```
Dim c As Range
For Each c In [A1:A100]
   If c.Value Like "*MS*" Then
      c.Value = "Microsoft"
      c.Interior.Color = RGB(255, 255, 0)
   Else
      c.Interior.Color = RGB(255, 255, 255)
   End If
Next
```

Ввод или считывание формулы в ячейку в формате A1

Свойство `Formula` объекта `Range` возвращает или устанавливает формулу в диапазон в формате **A1**. Например, в следующем примере первая инструкция вводит в ячейку **C1** формулу `=$A$1+$B$1`, а вторая — в ячейку **C2** формулу `=SIN(A2)^2`.

```
Range("C1").Formula = "=$A$1+$B$1"
Range("C2").Formula = "=SIN(A2)^2"
```

Ввод или считывание формулы в ячейку в формате R1C1

Свойство `FormulaR1C1` объекта `Range` возвращает формулу в формате **R1C1**. Например, следующая инструкция вводит в ячейку **B1** формулу `=2*R3C2` в **R1C1** формате или, что эквивалентно, формулу `=2*$B$3` в формате **A1**.

```
Range("B1").FormulaR1C1 = "=2*R3C2"
```

Ввод или считывание формулы локальной версии в ячейку в формате A1

Свойство `FormulaLocal` объекта `Range` возвращает формулу локальной версии в формате **A1**. Например, следующая инструкция вводит в ячейку **B2** формулу `=СУММ(C1:C4)`.

```
Range("B2"). FormulaLocal = "=СУММ(C1:C4)"
```

Ввод или считывание формулы локальной версии в ячейку в формате R1C1

Свойство `FormulaR1C1Local` объекта `Range` возвращает формулу локальной версии в формате **R1C1**. Например, следующая инструкция вводит в ячейку **B2** формулу `=СУММ(C1:C4)` в формате **R1C1**.

```
Range("B2").FormulaR1C1Local = "= СУММ(R1C3:R4C3)".
```

Ввод формулы массива в диапазон

Свойство `FormulaArray` объекта `Range` возвращает формулу диапазона в формате **A1**. В отличие от обыкновенной формулы рабочего листа, формула массива вводится на рабочем листе не нажатием клавиши <Enter>, а комбинацией клавиш <Ctrl>+<Shift>+<Enter>. Например, следующая инструкция вводит в диапазон **E1:E3** формулу `{=A1:A3*3}`:

```
Range("E1:E3").FormulaArray = "=A1:A3*3"
```

Ввод формулы массива локальной версии в диапазон

При вводе формулы массива с функциями рабочего листа локальной версии, формулу надо представить в формате **R1C1**, и вместо формулы локальной версии использовать формулу базовой версии. Например, следующая инструкция вводит в ячейку **D1** формулу `{=СУММ($A$1:$B$1*3)}`.

```
Range("D1").FormulaArray = "=SUM(R1C1:R1C2*3)"
```

Ввод формулы массива в диапазон с относительными ссылками на ячейки

Для ввода формулы с относительными ссылками на ячейки необходимо использовать относительную адресацию в формате **R1C1**. Например, ввод формулы `{=СУММ(A1:B1*3)}` в ячейку **D1** производится следующей инструкцией:

```
Range("D1").FormulaArray = "=SUM(RC[-3]:RC[-1]*3)"
```

Определение адреса ячейки

Свойство `Address` объекта `Range` возвращает адрес диапазона.

`Address(`*`RowAbsolute`*`, `*`ColumnAbsolute`*`, `*`ReferenceStyle`*`, `*`External`*`, `*`RelativeTo`*`)`, где:

- *`RowAbsolute`* — необязательный параметр, принимающий логические значения. Если значение параметра равно `True` или параметр опущен, то возвращается абсолютная ссылка на строку;
- *`ColumnAbsolute`* — необязательный параметр, принимающий логические значения. Если его значение равно `True` или параметр опущен, то возвращается абсолютная ссылка на столбец;
- *`ReferenceStyle`* — необязательный параметр. Допустимы два значения — `xlA1` и `xlR1C1`, если `xlA1` или `xlR1C1` опущены, то возвращается ссылка в формате **A1**;
- *`External`* — необязательный параметр, принимающий логические значения и определяющий, является ли ссылка внешней;
- *`RelativeTo`* — необязательный параметр. В случае если значения параметров *`rowAbsolute`* и *`columnAbsolute`* равны `False`, а *`referenceStyle`* — `xlR1C1`, то данный параметр определяет начальную ячейку диапазона, относительно которой производится адресация.

Следующий код (листинг 7.44), обрабатывающий событие `SelectionChange` объекта `Worksheet`, демонстрирует возвращаемые свойством `Address` значения при различных установках его параметров. Например, если на рабочем листе будет выбрана ячейка **A1**, то на экране отобразится окно со следующим сообщением:

```
$A$1
$A1
R1C1
R[-1]C[-1]
```

Листинг 7.44. Свойство `Address`. Модуль рабочего листа

```
Private Sub Worksheet_SelectionChange(ByVal Target As Range)
   MsgBox Target.Address() & vbCr & _
          Target.Address(RowAbsolute:=False) & vbCr & _
          Target.Address(ReferenceStyle:=xlR1C1) & vbCr & _
          Target.Address(ReferenceStyle:=xlR1C1, _
                         RowAbsolute:=False, _
                         ColumnAbsolute:=False, _
                         RelativeTo:=Worksheets(1).Cells(2, 2))
End Sub
```

Управление стилем границы диапазона и объектами *Border*

Свойство `Borders` объекта `Range` возвращает семейство `Borders`, элементы которого инкапсулируют данные об одной из граничных или диагональных линий данного диапазона. Допустимыми значениями индекса семейства `Borders` могут быть следующие константы `XlBordersIndex`: `xlDiagonalDown`, `xlDiagonalUp`, `xlEdgeBottom`, `xlEdgeLeft`, `xlEdgeRight`, `xlEdgeTop`, `xlInsideHorizontal` и `xlInsideVertical`. Каждая из этих границ представляет собой объект `Border`, свойства которого перечислены в табл. 7.16.

***Таблица 7.16.** Свойства объекта `Border`*

Свойство	Описание
`Color`	Цвет границы, заданный при помощи RGB-модели
`ColorIndex`	Цвет границы, заданный индексом соответствующего элемента цветовой палетки
`LineStyle`	Задает стиль границы. Допустимыми значениями являются следующие константы `XlLineStyle`: `xlContinuous`, `xlDash`, `xlDashDot`, `xlDashDotDot`, `xlDot`, `xlDouble`, `xlSlantDashDot` и `xlLineStyleNone`
`Weight`	Устанавливает толщину границы. Допустимыми значениями являются следующие константы `XlBorderWeight`: `xlHairline`, `xlThin`, `xlMedium` и `xlThick`

Например, следующий код (листинг 7.45) задает верхнюю границу диапазона **A2:C2** в виде толстой линии красного цвета, а его нижнюю границу — в виде зеленой пунктирной линии средней толщины.

Листинг 7.45. Конструирование границы специфицированного диапазона

```
Sub DemoBorders()
   Dim rgn As Range
   Set rgn = Range("A2:C2")
   With rgn.Borders(xlEdgeTop)
      .LineStyle = xlContinuous
      .Weight = xlThick
      .Color = RGB(255, 0, 0)
   End With
   With rgn.Borders(xlEdgeBottom)
      .LineStyle = xlDash
      .Weight = xlMedium
```

```
        .Color = RGB(0, 255, 0)
    End With
End Sub
```

Если все компоненты границы имеют одни и те же параметры, то для установки их значения можно воспользоваться не элементами, а всем семейством `Borders`, как это, например, делается в следующей инструкции для создания границы синего цвета у выделенной области.

```
Selection.Borders.Color = RGB(0, 0, 255)
```

Функции *RGB* и *QBColor*

Коды цветов в VBA часто задаются числами в форме шестнадцатеричной системы счисления. Вместо прямого указания шестнадцатеричного кода цвета, довольно часто цвет удобнее задавать, используя функции `RGB` и `QBColor`. Функция `RGB` позволяет получить любой цвет смешением красного, зеленого и синего компонентов различной интенсивности.

`RGB(`*`Red`*`, `*`Green`*`, `*`Blue`*`)`, где:

- *`Red`* — целое число из диапазона от 0 до 255, указывающее красный компонент цвета;
- *`Green`* — целое число из диапазона от 0 до 255, указывающее зеленый компонент цвета;
- *`Blue`* — целое число из диапазона от 0 до 255, указывающее синий компонент цвета.

В табл. 7.17 приведены значения параметров функции `RGB` для получения стандартных цветов.

Таблица 7.17. *Значения параметров функции `RGB` для получения стандартных цветов*

Цвет	Red	Green	Blue
Черный	0	0	0
Синий	0	0	255
Зеленый	0	255	0
Голубой	0	255	255
Красный	255	0	0
Розовый	255	0	255
Желтый	255	255	0
Белый	255	255	255

Функция `QBColor` возвращает шестнадцать основных цветов в зависимости от значения параметра (табл. 7.18).

```
QBColor(color)
```

Здесь параметр *`color`* принимает целые числа из диапазона от 0 до 15.

Таблица 7.18. *Соответствие между цветами и значением параметра функции* `QBColor`

Число	Цвет	Число	Цвет
0	Черный	8	Серый
1	Синий	9	Светло-синий
2	Зеленый	10	Светло-зеленый
3	Голубой	11	Светло-голубой
4	Красный	12	Светло-красный
5	Розовый	13	Светло-розовый
6	Желтый	14	Светло-желтый
7	Белый	15	Насыщенный белый

Доступ к отдельным ячейкам диапазона

Свойство `Cells` объекта `Range`, использованное без индексов, возвращает все ячейки диапазона, а с индексами — конкретную ячейку, специфицированную либо ее номером (один параметр), либо местоположением (два параметра).

Например, в следующем коде (листинг 7.46) в диапазоне **B1:C3** все положительные значения заменяются на 1, а отрицательные — на −1.

Листинг 7.46. Все ячейки диапазона

```
Dim c As Range
For Each c In Range("B1:C3").Cells
  If c.Value > 0 Then
     c.Value = 1
  ElseIf c.Value < 0 Then
     c.Value = -1
  End If
Next
```

Ту же задачу можно решить, используя свойство `Cells` с двумя параметрами, возвращающее ячейку, стоящую на пересечении указанной строчки и столбца. При этом надо помнить, что свойство `Cells` задает относительное расположение ячеек по отношению к диапазону (листинг 7.47).

Листинг 7.47. Относительное местоположение ячеек

```
Dim i As Integer
Dim j As Integer
For i = 1 To Range("B1:C3").Columns.Count
   For j = 1 To Range("B1:C3").Columns.Count
      If Range("B1:C3").Cells(i, j).Value > 0 Then
         Range("B1:C3").Cells(i, j).Value = 1
      ElseIf Range("B1:C3").Cells(i, j).Value < 0 Then
         Range("B1:C3").Cells(i, j).Value = -1
      End If
   Next
Next
```

Если требуется задать абсолютное местоположение ячеек, то надо воспользоваться свойством `Cells` рабочего листа, например, как это делается для той же самой задачи в листинге 7.48.

Листинг 7.48. Абсолютное местоположение ячеек

```
Dim i As Integer
Dim j As Integer
For i = 2 To 3
   For j = 1 To 3
      If Cells(i, j).Value > 0 Then
         Cells(i, j).Value = 1
      ElseIf Cells(i, j).Value < 0 Then
         Cells(i, j).Value = -1
      End If
   Next
Next
```

Выбор элементов на рабочем листе или в книге

Выбор элемента, например, диапазона на рабочем листе производится методом `Select`. Допустим, для того чтобы очистить содержимое ячейки `A1`,

ее надо выбрать, а затем к полученному объекту `Selection` применить метод `ClearContents`.

```
Range("A1").Select
Selection.ClearContents
```

Конечно, тот же самый эффект можно достичь, применив метод `ClearContents` непосредственно к объекту `Range("A1")`

```
Range("A1").ClearContents
```

Затем, не изменяя текущей активной ячейки, например, при помощи метода `Offset` переместиться к другой ячейке (в данном случае ячейке `B2`) и произвести над ней какие-либо действия, например, очистить содержимое и у нее.

```
Range("A1").Offset(1,1).ClearContents
```

Метод `Select` позволяет выбрать не только одну ячейку, но и целый диапазон. Например, первая из приводимых инструкций выбирает диапазон `A1:B2`, вторая — ячейки `A1, B2, D9`, а третья — диапазон `B2:C4` и ячейки `A1, D9`.

```
Range("A1:B2").Select
Range("A1,B2,D9").Select
Range("A1,B2:C4,D9").Select
```

Метод `Activate` позволяет не только выбирать, но и активизировать указанный объект. Например, первая из приводимых инструкций активизирует рабочую книгу, в которой в данный момент выполняется макрос, вторая — окно с рабочей книгой `Отчет.xls`, а третья — рабочий лист **Май**.

```
ThisWorkbook.Activate
Windows("Отчет.xls").Activate
Sheets("Май").Activate
```

Свойство `ActiveCell` возвращает активную ячейку. Например, в следующем коде сначала определяется активная ячейка (например, ей будет `A1`), затем методом `Offset` возвращается ссылка на ячейку, которая находится на одну строку ниже и на три столбца правее активной, и эта ячейка затем выбирается методом `Select` (в данном случае это `E2`).

```
ActiveCell.Offset(1, 3).Select
```

Если имеется несколько ячеек, имена которых образуют нумерованную последовательность, то для работы с ними можно воспользоваться оператором цикла. Например, в следующем коде ячейкам с именами `c_1`, `c_2` и `c_3` присваиваются в цикле значения, соответствующие их индексам.

```
Sub SetValues()
   Dim i As Integer
   Dim cellName As String
   For i = 1 To 3
      cellName = "c_" & i
      Range([cellName]).Value = i
```

```
   Next
End Sub
```

Для того чтобы выбрать первую ячейку столбца/строки, в которой находится активная ячейка, можно воспользоваться следующими инструкциями:

```
ActiveCell.Offset(0, -ActiveCell.Column + 1).Select
ActiveCell.Offset(-ActiveCell.Row + 1,0).Select
```

Для того чтобы выбрать диапазон от указанной ячейки (например, активной) вниз/вверх вдоль столбца или влево/вправо вдоль строки до первой пустой ячейки, подойдут следующие инструкции:

```
Range(ActiveCell, ActiveCell.End(xlDown)).Select
Range(ActiveCell, ActiveCell.End(xlUp)).Select
Range(ActiveCell, ActiveCell.End(xlToLeft)).Select
Range(ActiveCell, ActiveCell.End(xlToRight)).Select
```

Для того чтобы выбрать диапазон ячеек, начиная от активной ячейки до другой, находящейся на заданном расстоянии от активной, можно применить следующую инструкцию:

```
Range(ActiveCell, ActiveCell.Offset(3, 4)).Select
```

Для того чтобы выбрать первую пустую ячейку в столбце/строке (в данном случае столбец **A**/первая строка), воспользуйтесь следующими инструкциями:

```
Range("A1").End(xlDown).Offset(1, 0).Select
Range("A1").End(xlToRight).Offset(0,1).Select
```

Для изменения размеров выбранного диапазона применяется метод `Resize`, который возвращает диапазон, имеющий указанные размеры, и с левой верхней вершиной, совпадающей с соответствующей вершиной искомого диапазона. Например, если первоначально выбран диапазон `B2:C4`, то следующая инструкция выбирает диапазон `B2:F5`.

```
Selection.Resize(4, 5).Select
```

Как проверить, является ли выбранный объект диапазоном

Функция `TypeName` (листинг 7.49) возвращает имя типа значения ее параметра. Поэтому, если надо проверить, является ли выбранный объект диапазоном, просто надо воспользоваться функцией `TypeName`.

Листинг 7.49. Функция, проверяющая является ли выбранный объект диапазоном

```
Function IsRange() As Boolean
   If TypeName(Selection) = "Range" Then
      IsRange = True
```

```
   Else
      IsRange = False
   End If
End Function
```

Объект *Characters* (как форматировать часть содержимого ячейки)

Свойство `Characters` объекта `Range` или `Selection` возвращает объект `Characters`, представляющий собой строку указанной длины от указанного символа. Часто применяется, когда надо форматировать содержание не всей строки, а только ее часть.

`Characters(`*`Start`*`, `*`Length`*`)`, где:

- *`Start`* — необязательный параметр, задающий номер первого возвращаемого символа из данной строки;
- *`Length`* — необязательный параметр, указывающий число возвращаемых символов.

В следующем примере (листинг 7.50) в ячейку **A1** выводится строка `"Андрей Гарнаев"`, причем первая часть этой строки `"Андрей"` выводится полужирным шрифтом зеленого цвета высотой 16, а вторая ее часть `"Гарнаев"` — курсивным шрифтом красного цвета высотой 14. Кроме того, в рабочий лист вставляется автофигура, в которой размещается текстовое поле, в которое выводится тот же текст также отформатированный по словам (рис. 7.5).

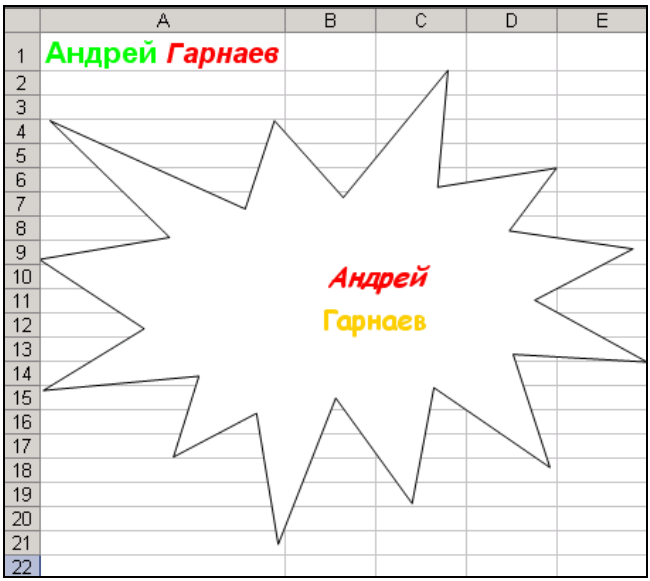


Рис. 7.5. Форматирование части содержимого ячейки

Листинг 7.50. Форматирование части содержимого ячейки

```
Sub DrawText()
   With Range("A1")
      .Value = "Андрей Гарнаев"
      .Characters(1, 6).Font.Bold = True
      .Characters(1, 6).Font.Size = 16
      .Characters(1, 6).Font.Color = RGB(0, 255, 0)
      .Characters(8, 7).Font.Italic = True
      .Characters(8, 7).Font.Size = 14
      .Characters(8, 7).Font.Color = RGB(255, 0, 0)
   End With

   ActiveSheet.Shapes.AddShape(msoShapeExplosion1, _
         Range("A2").Left, Range("A2").Top, _
         Range("F20").Left, Range("F20").Top).Select
   ActiveSheet.Shapes.AddTextbox(msoTextOrientationHorizontal, _
         Range("B9:C13").Left, Range("B9:C13").Top, _
         Range("B9:C13").Width, Range("B9:C13").Height).Select
   Selection.HorizontalAlignment = xlCenter
   Selection.VerticalAlignment = xlCenter
   Selection.Characters.Text = "Àíäðåé Ãàðíàåâ"
   Selection.ShapeRange.Line.Visible = msoFalse
   Selection.Characters.Text = " Андрей Гарнаев "
   With Selection.Characters(Start:=1, Length:=7).Font
      .Name = "Comic Sans MS"
      .FontStyle = "полужирный курсив"
      .Size = 14
      .ColorIndex = 3
   End With
   With Selection.Characters(Start:=8, Length:=7).Font
      .Name = "Comic Sans MS"
      .FontStyle = "полужирный"
      .Size = 14
     .ColorIndex = 44
   End With
   Range("A1").Select
End Sub
```

Объект *Comment* (создание комментариев к диапазону)

Свойство `Comment` объекта `Range` возвращает объект `Comment` (*примечание*), которое при отображении на экране связано с правым верхним углом диапазона. Вручную на рабочем листе примечание добавляется к диапазону выбором команды **Вставка | Примечание**.

В коде примечание можно создать методом `AddComment` объекта `Range`.

`AddComment(`*`Text`*`)`

Здесь *`Text`* — текст, вводимый в примечание. Объект `Comment` является элементом семейства `Comments`. В табл. 7.19 перечислены свойства объекта `Comment`, а в табл. 7.20 — его методы.

Таблица 7.19. *Свойства объекта* `Comment`

Свойство	Описание
`Author`	Возвращает автора примечания
`Shape`	Возвращает объект `Shape`, задающий форму примечания
`Parent`	Возвращает ссылку на диапазон, для которого примечание было создано
`Visible`	Устанавливает, отображается ли примечание при активизации диапазона

Таблица 7.20. *Методы объекта* `Comment`

Метод	Описание
`Delete`	Удаляет примечания
`Next`	Возвращает следующее примечание
`Previous`	Возвращает предыдущее примечание
`Text`	Возвращает текст, отображаемый в примечании

Итак, метод `Text` задает текст, выводимый в примечании.

`Text(`*`Text`*`, `*`Start`*`, `*`Overwrite`*`)`, где:

- *`Text`* — необязательный параметр, который задает текст, выводимый в качестве примечания;
- *`Start`* — необязательный параметр, который определяет, с какого символа вводится текст в уже существующее примечание. Если параметр опущен, то из примечания удаляется весь ранее введенный текст;
- *`Overwrite`* — необязательный параметр, принимающий логические значения. Если он равен `True`, то вводимый текст записывается вместо уже существующего, а если `False`, то добавляется к уже существующему.

В качестве примера рассмотрим процедуру обработки события `Open` объекта `Workbook`, который при открытии рабочей книги в ячейке **G4** первого рабочего листа создает комментарий (листинг 7.51, *а*). При этом, если в данной ячейке уже существовал комментарий, то он удаляется и затем создается по-новому. Процедура же обработки события `BeforeDoubleClick` объекта `Worksheet` обеспечивает отображение или скрытие примечания при двойном щелчке на листе (листинг 7.51, *б*).

Листинг 7.51, *а*. Создание примечания при открытии книги. Модуль `ЭтаКнига`

```
Private Sub Workbook_Open()
   Set r = Worksheets(1).Range("G4").Comment
   If Not IsObject(r) Then
      Worksheets(1).Range("G4").Comment.Delete
   End If
   Worksheets(1).Range("G4").AddComment Text:="Очень важно!"
End Sub
```

Листинг 7.51, *б*. Отображение или скрытие примечания при двойном щелчке на листе. Модуль рабочего листа

```
Private Sub Worksheet_BeforeDoubleClick(ByVal Target As Range, _
                                        Cancel As Boolean)
   If Range("G4").Comment.Visible Then
      Range("G4").Comment.Visible = False
   Else
      Range("G4").Comment.Visible = True
   End If
End Sub
```

Определение текущего диапазона

Свойство `CurrentRegion` объекта `Range` возвращает *текущий диапазон* (current region). Текущий диапазон — это максимальный диапазон, который окружен любой комбинацией пустых строк и столбцов, ячейки которого не пусты, а сам он содержит данный диапазон.

Например, в диапазоне **A1:E12** имеется таблица, которая по столбцу **F** и 13-ой строке окружена пустыми ячейками. Тогда следующая инструкция вызовет выделение всей таблицы, т. е. диапазон **A1:E12**.

```
Cells(1, 1).CurrentRegion.Select
```

Немного усложним задачу. Пусть у этой таблицы первая строка отведена под заголовок. Тогда для того чтобы выделить всю таблицу без заголовка, достаточно применить следующие инструкции:

```
Dim r As Range
Set r = Cells(1, 1).CurrentRegion
r.Offset(1, 0).Resize(r.Rows.Count - 1, r.Columns.Count).Select
```

В следующем примере переменной `y` присваивается значение, равное числу строк в текущем диапазоне, содержащем ячейку **A1**.

```
y = Range("A1").CurrentRegion.Rows.Count
```

Нахождение крайней ячейки диапазона в указанном направлении

Свойство `End` объекта `Range` возвращает крайнюю ячейку в указанном направлении в диапазоне, содержащем заданный диапазон.

```
End(Direction)
```

Здесь допустимыми значениями параметра *`Direction`* может быть одна из следующих констант `XlDirection`: `xlToLeft`, `xlToRight`, `xlUp` и `xlDown`.

Например, в следующем примере строка "`x`" будет введена в ячейку **A4**, а "`y`" — в ячейку **B2**:

```
Range("A1:B4").Value = "Тест"
Range("A1:B4").Select
Selection.End(xlDown).Value = "x"
Range("A2").End(xlToRight).Value = "y"
```

Нахождение строки и столбца, содержащих данную ячейку

Свойства `EntireColumn` и `EntireRow` объекта `Range` возвращают столбец и строку, содержащие данную ячейку. Например, следующий код очищает всю строку, содержащую данную ячейку.

```
ActiveCell.EntireRow.Clear
```

Объект *Hyperlink* (задание гиперссылки)

Свойство `Hyperlinks` объекта `Range` возвращает семейство `Hyperlinks`, состоящее из объектов `Hyperlink`, инкапсулирующих в себе информацию о гиперссылках.

У семейства `Hyperlinks` наиболее важными свойствами являются свойства `Count` и `Item`, возвращающие число элементов и конкретный элемент семейства. Кроме того, в этом семействе имеются два метода: метод `Delete`, который удаляет гиперссылку, и метод `Add`, который создает новый элемент семейства.

`Add(Anchor, Address, SubAddress, ScreenTip, TextToDisplay)`, где:

- *`Anchor`* — обязательный параметр, идентифицирующий местоположение гиперссылки;
- *`Address`* — обязательный параметр, задающий адрес гиперссылки;
- *`SubAddress`* — необязательный параметр, определяющий локальный адрес по отношению к адресу документа;
- *`ScreenTip`* — необязательный параметр, задающий текст всплывающей подсказки;
- *`TextToDisplay`* — необязательный параметр, задающий текст гиперссылки.

Основные свойства объекта `Hyperlink` перечислены в табл. 7.21, а методы — в табл. 7.22.

Таблица 7.21. *Свойства объекта* `Hyperlink`

Свойство	Описание
`Address`	Возвращает адрес документа, в который происходит переход
`EmailSubject`	Возвращает строку, отображаемую в разделе **Тема сообщения**, при отсылке электронной почты
`Name`	Возвращает имя объекта
`Parent`	Возвращает ссылку на объект (диапазон), в котором расположена гиперссылка
`Range`	Возвращает ссылку на диапазон, в котором расположена гиперссылка
`ScreenTip`	Возвращает текст всплывающей подсказки
`Shape`	Возвращает объект `Shape`, к которому присоединена гиперссылка
`SubAddress`	Возвращает локальный адрес по отношению к адресу документа
`TextToDisplay`	Возвращает текст гиперссылки

Таблица 7.22. *Методы объекта* `Hyperlink`

Метод	Описание
`AddToFavorites`	Добавляет гиперссылку в раздел **Избранные** браузера
`CreateNewDocument`	Открывает новый документ, связанный с указанной гиперссылкой.
`Delete`	Удаляет гиперссылку

Таблица 7.22 *(окончание)*

Метод	Описание
`Follow`	Отображает указанный документ, если он уже загружен. В противном случае производит его загрузку и отображение. `Follow(NewWindow, AddHistory, ExtraInfo, Method, HeaderInfo)`, где: *`NewWindow`* — необязательный параметр, если он принимает значение `True`, то документ отображается в новом окне, если же его значение равно `False`, то в тоже самом; *`AddHistory`* — не используется, зарезервирован для будущих версий MS Excel; *`ExtraInfo`* — необязательный параметр, содержащий дополнительную информацию, используемую для HTTP-протокола; *`Method`* — необязательный параметр, специфицирующий то, как передается информация из параметра *`ExtraInfo`*; *`HeaderInfo`* — необязательный параметр, содержащий заголовочную информацию, передаваемую при HTTP-запросе

Например, следующая инструкция в ячейке **A1** создает гиперссылку `MS` со всплывающей подсказкой `Microsoft` для открытия Web-страницы **http://microsoft.com**.

```
With Worksheets(1)
    .Hyperlinks.Add Anchor:=.Range("A1"), _
                    Address:="http://microsoft.com", _
                    ScreenTip:="Microsoft", _
                    TextToDisplay:="MS"
End With
```

Объект *Font* (задание шрифта)

Свойство `Font` объекта `Range` возвращает объект `Font`, представляющий собой шрифт. В табл. 7.23 приведены свойства объекта `Font`.

Таблица 7.23. *Свойства объекта* `Font`

Свойство	Описание
`Bold`	Определяет, является ли шрифт полужирным
`Color`	Задает цвет шрифта в соответствии с RGB-моделью
`ColorIndex`	Задает индексированный цвет в соответствии с текущей палеткой цветов

Таблица 7.23 *(окончание)*

Свойство	Описание
FontStyle	Задает стиль шрифта, указанный в словесной форме. Допустимы значения: Regular (обычный), Bold (полужирный), Italic (курсив), Bold Italic (полужирный курсив)
Italic	Определяет, является ли шрифт курсивным
Name	Строка, указывающая имя шрифта, например "Arial Cyr"
Size	Размер шрифта
Strikethrough	Устанавливает, имеется ли линия по центру, как будто текст перечеркнут
Superscript	Устанавливает, используется ли текст как верхний индекс
Subscript	Устанавливает, используется ли текст как нижний индекс
Underline	Задает тип подчеркивания. Допустимы значения: xlNone (нет подчеркивания), xlSingle (одинарное, по значению), xlDouble (двойное, по значению), xlSingleAccounting (одинарное, по ячейке), xlDoubleAccounting (двойное, по ячейке)

Например, в следующем примере устанавливается для диапазона **A1:B2** полужирный шрифт красного цвета с высотой символов 14.

```
With Range("A1:B2").Font
    .Size = 14
    .Bold = True
    .Color = RGB(255, 0, 0)
End With
```

Объект *Interior* (заливка диапазона)

Свойство Interior объекта Range возвращает объект Interior, инкапсулирующий данные о заливке диапазона. В табл. 7.24 приведены свойства объекта Interior.

Таблица 7.24. *Свойства объекта* Interior

Свойство	Описание
Color	Задает цвет заливки в соответствии с RGB-моделью
ColorIndex	Задает индексированный цвет заливки в соответствии с текущей палеткой цветов

Таблица 7.24 *(окончание)*

Свойство	Описание
`Pattern`	Задает узор заливки. Допустимыми являются следующие значения: `xlPatternAutomatic`, `xlPatternChecker`, `xlPatternCrissCross`, `xlPatternDown`, `xlPatternGray16`, `xlPatternGray25`, `xlPatternGray50`, `xlPatternGray75`, `xlPatternGray8`, `xlPatternGrid`, `xlPatternHorizontal`, `xlPatternLightDown`, `xlPatternLightHorizontal`, `xlPatternLightUp`, `xlPatternLightVertical`, `xlPatternNone`, `xlPatternSemiGray75`, `xlPatternSolid`, `xlPatternUp`, `xlPatternVertical`
`PatternColor`	Задает цвет узора в соответствии с RGB-моделью
`PatternColorIndex`	Задает индексированный цвет узора в соответствии с текущей палеткой цветов

Например, в следующем примере (листинг 7.52) устанавливается красная заливка диапазона **A1:D5** с синим клетчатым узором.

Листинг 7.52. Заливка диапазона

```
With Range("A1:D5")
   .Interior.Color = RGB(255, 0, 0)
   .Interior.Pattern = xlPatternChecker
   .Interior.PatternColor = RGB(0, 0, 255)
End With
```

В следующем коде (листинг 7.53) выводится таблица цветов и соответствующих значений свойства `ColorIndex` (рис. 7.6).

Листинг 7.53. Таблица цветов

```
Sub ListOfColorIndex()
   Dim c As Long
   Range("A1").Value = "Цвет"
   Range("B1").Value = "Значение свойства ColorIndex"
   Range("A2").Select
   For c = 1 To 56
      With ActiveCell.Interior
         .ColorIndex = c
         .Pattern = xlSolid
```

```
            .PatternColorIndex = xlAutomatic
        End With
        ActiveCell.Offset(0, 1).Value = c
        ActiveCell.Offset(1, 0).Activate
    Next
    Range("A1").Select
    ActiveWindow.ScrollRow = 1
End Sub
```

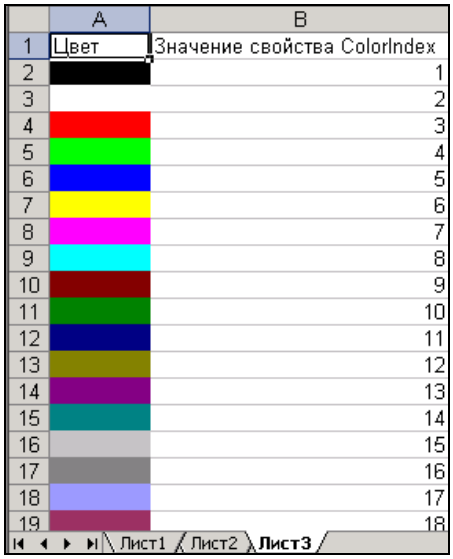


Рис. 7.6. Таблица цветов

В качестве еще одной иллюстрации приведем код, который изменяет цвет выбранного диапазона с серого на заданный.

```
Private Sub Worksheet_SelectionChange(ByVal Target As Range)
    Cells.Interior.ColorIndex = xlColorIndexNone
    Target.Interior.Color = RGB(243, 167, 242)
End Sub
```

Разрешение редактирования содержимого ячеек на защищенном рабочем листе

Свойство `Locked` объекта `Range`, установленное равным значению `False`, обеспечивает пользователю возможность редактирования содержимого ячеек даже на защищенном рабочем листе. Например, следующая процедура (листинг 7.54), обрабатывающая событие `Open` объекта `Workbook`, вводит в ячейку

C1 формулу `=A1+B1` и защищает по паролю `stop` все ячейки листа, кроме ячеек **A1**и **B1**, в которые пользователь может вводить любые значения.

Листинг 7.54. Разрешение редактирования содержимого ячеек на защищенном рабочем листе. Модуль ЭтаКнига

```
Private Sub Workbook_Open()
   On Error Resume Next
   Worksheets(1).Range("C1").Formula = "=A1+B1"
   Worksheets(1).Range("A1:B1").Locked = False
   Worksheets(1).Protect "stop"
End Sub
```

Установка числового формата

Свойство `NumberFormat` объекта `Range` устанавливает числовой формат. Например, следующие инструкции (листинг 7.55):

- в ячейке **A1** устанавливают общий формат;
- в ячейке **A2** — числовой формат, отображающий три знака после десятичной точки, например `12.000`;
- в ячейке **A3** — формат времени с двоеточием в качестве разделителя и по два знака, отведенных под часы, минуты и секунды, например `02:12:55`;
- в ячейке **A4** — формат даты, причем два знака отводится под день, три буквы — под месяц и четыре цифры — под год, например `01 июл 2002`.

Листинг 7.55. Установка числового формата

```
Range("A1").NumberFormat = "General"
Range("A2").NumberFormat = "0.000"
Range("A3").NumberFormat = "hh:mm:ss"
Range("A4").NumberFormat = "d mmm yyyy"
```

Нахождение диапазона, сдвинутого относительно данного на указанное число строк и столбцов

Свойство `Offset` объекта `Range` возвращает объект `Range`, который сдвинут по отношению к данному на указанное число строк и столбцов.

`Offset(`*`RowOffset`*`, `*`ColumnOffset`*`)`, где:

- *`RowOffset`* — необязательный параметр, указывающий сдвиг по строкам;
- *`ColumnOffset`* — необязательный параметр, указывающий сдвиг по столбцам.

Например, в следующем коде в активную ячейку, допустим **A1**, вводится значение `3`, в ячейку, сдвинутую от нее на один столбец (в данном случае **B1**), вводится значение `4`, а в ячейку, сдвинутую еще на один столбец (в данном случае **C1**), вводится формула, находящая сумму содержимого двух первых ячеек.

```
ActiveCell.Value = 3
ActiveCell.Offset(0, 1).Value = 4
ActiveCell.Offset(0, 2).FormulaR1C1 = "=R[0]C[-2]+R[0]C[-1]"
```

Задание угла, под которым выводится текст в диапазоне

Свойство `Orientation` объекта `Range` устанавливает угол, под которым выводится текст в диапазоне. Допустимыми значениями являются либо угол поворота текста в градусах от −90° до 90°, либо одна из следующих констант:

- `xlDownward` — выравнивание по левому краю сверху вниз, соответствует углу 90°;
- `xlHorizontal` — выравнивание по горизонтали, соответствует нулевому углу;
- `xlUpward` — выравнивание по правому краю снизу вверх, соответствует углу 90°;
- `xlVertical` — выравнивание по вертикали, нет соответствия в градусах.

Например, в следующем коде (листинг 7.56) в ячейке **A1** текст выводится под углом 45°, а в ячейке **B1** — под углом −45° (рис. 7.7).

Листинг 7.56. Задание угла вывода текста

```
Range("A1:B1").Font.Size = 16
Range("A1").Orientation = 45
Range("A1").Value = "Вверх"
Range("B1").Orientation = -45
Range("B1").Value = "Вниз"
```

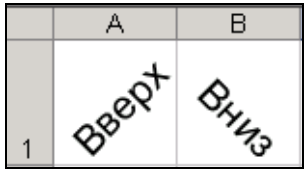


Рис. 7.7. Задание угла вывода текста

Переопределение размеров диапазона

Свойство `Resize` объекта `Range` возвращает диапазон с измененными размерами.

`Resize(`*`RowSize`*`, `*`ColumnSize`*`)`, где:

- *`RowSize`* — необязательный параметр, указывающий новое число строк;
- *`ColumnSize`* — необязательный параметр, указывающий новое число столбцов.

Например, предположим, что на рабочем листе имеется таблица с левым верхним углом, расположенным в ячейке **A1**, причем первая строка таблицы является заголовком. Тогда следующие инструкции создают красную заливку для заголовка таблицы и желтую — для ее тела.

```
Dim tbl As Range
Set tbl = Range("A1").CurrentRegion
Set tbl = tbl.Resize(1, tbl.Columns.Count)
tbl.Interior.Color = RGB(255, 0, 0)
Set tbl = Range("A1").CurrentRegion
Set tbl = tbl.Offset(1, 0).Resize(tbl.Rows.Count - 1, tbl.Columns.Count)
tbl.Interior.Color = RGB(255, 255, 0)
```

Методы объекта *Range*

Объект `Range` обладает большой коллекцией методов, предоставляющих в распоряжение разработчика возможность программировать целый спектр действий от копирования диапазона в буфер обмена до нахождения корня нелинейного уравнения. Перечислим основные методы этого объекта.

`Activate`	`AddComment`	`AutoFill`
`AutoFit`	`BorderAround`	`Clear`
`ClearComments`	`ClearComments`	`ClearContents`
`ClearFormats`	`ClearNotes`	`Copy`
`CopyPicture`	`Cut`	`DataSeries`
`Delete`	`FillDown`	`FillLeft`
`FillRight`	`FillUp`	`Find`
`FindNext`	`FindPrevious`	`FunctionWizard`
`GoalSeek`	`Insert`	`PasteSpecial`
`Replace`	`Select`	`Show`

Активизация и выбор диапазона

Метод `Activate` объекта `Range` активизирует диапазон, а метод `Select` выбирает диапазон, т. е. возвращает объект `Selection`. Например, в следующем коде сначала активизируется ячейка **A2**, затем в активную ячейку

вводится число 1, после чего выбирается диапазон **A3:A4** и в выбранный диапазон вводится число 3.

```
Range("A2").Activate
ActiveCell.Value =  1
Range("A3:A4").Select
Selection.Value = 3
```

Вставка и удаление комментариев в диапазон

Метод AddComment объекта Range вставляет комментарий в диапазон, а метод ClearComments его удаляет. Например, следующий код (листинг 7.57) сканирует диапазон **A1:A100** и в случае, если в очередную ячейку входит строка BHV, сначала эта ячейка очищается от комментариев, а затем в нее вставляется комментарий Книга издательства BHV.

Листинг 7.57. Вставка и удаление комментариев в диапазон

```
Sub DemoComments()
   Dim c As Range
   For Each c In Range("A1:A100")
      If c.Value Like "*BHV*" Then
         c.ClearComments
         c.AddComment "Книга издательства BHV"
      End If
   Next
End Sub
```

Заполнение диапазона прогрессией

Метод DataSeries объекта Range заполняет диапазон прогрессией. Метод DataSeries программирует выполнение команды **Правка** | **Заполнить** | **Прогрессия**.

DataSeries(*RowCol*, *Type*, *Date*, *Step*, *Stop*, *Trend*), где:

- *RowCol* — необязательный параметр, определяющий направление, в котором создается прогрессия. Допустимые значения: xlRows (вдоль строк), xlColumns (вдоль столбцов). Если параметр опущен, то для задания направления используются размеры данного диапазона;
- *Type* — необязательный параметр, определяющий тип прогрессии. Допустимыми значениями являются следующие константы XlDataSeriesType: xlDataSeriesLinear (линейная, используется по умолчанию), xlGrowth (геометрическая), xlChronological (даты), xlAutoFill (автозаполнение);

- *Date* — необязательный параметр, задающий тип последовательности дат, если параметр *Type* принимает значение `xlChronological`. Допустимыми значениями являются следующие константы `XlDataSeriesDate`: `xlDay` (дни, используется по умолчанию), `xlWeekday` (дни недели), `xlMonth` (месяцы), `xlYear` (годы);
- *Step* — необязательный параметр, определяющий шаг изменения прогрессии. По умолчанию полагается равным 1;
- *Stop* — необязательный параметр, задающий предельное значение прогрессии. По умолчанию строится прогрессия во всем выделенном диапазоне;
- *Trend* — необязательный параметр, принимает логические значения. Если значение параметра равно `True`, то создается арифметическая или геометрическая прогрессия, а если `False`, то создается список.

Например, следующие инструкции (листинг 7.58, *а*) выведут членов арифметической прогрессии, шаг которой равен 2, 0 является ее первым членом, а 10 — последним, т. е. в диапазон **A1:A6** будут выведены следующие числа: 0, 2, 4, 6, 8 и 10 (рис. 7.8).

Листинг 7.58, *а*. Прогрессия

```
Range("A1").Value = 0
Range("A1").DataSeries Rowcol:=xlColumns, _
                       Type:=xlDataSeriesLinear, _
                       Step:=2, _
                       Stop:=10
```

	A	B	C
1	0	1	01.01.2003
2	2	3	01.02.2003
3	4	9	01.03.2003
4	6	27	01.04.2003
5	8	81	
6	10		

Рис. 7.8. Прогрессии

Следующие инструкции (листинг 7.58, *б*) введут в диапазон **B1:B5** члены геометрической прогрессии с коэффициентом 3, первый член которой — 1. Таким образом, в диапазон **B1:B5** будут выведены значения 1, 3, 9, 27 и 81 (рис. 7.8).

Листинг 7.58, *б*. Прогрессия

```
Range("B1").Value = 1
Range("B1:B5").DataSeries Rowcol:=xlColumns, _
```

```
                    Type:=xlGrowth, _
                    Step:=3
```

Следующие инструкции (листинг 7.58, *в*) выведут в диапазон **C1:C4** последовательность дат, члены которой различаются ровно на месяц, т. е. значения `01.01.2003`, `01.02.2003`, `01.03.2003`, `01.04.2003` (рис. 7.8).

Листинг 7.58, *в*. Прогрессия

```
Range("C1").Value = "1/01/03"
Range("C1:C4").DataSeries Rowcol:=xlColumns, _
                          Type:=xlChronological, _
                          Date:=xlMonth
```

Автозаполнение ячеек диапазона элементами последовательности

Метод `AutoFill` объекта `Range` производит автозаполнение ячеек диапазона элементами последовательности. Метод `AutoFill` отличается от метода `DataSeries` тем, что явно указывается диапазон, в котором будет располагаться прогрессия. Метод `AutoFill` программирует выполнение копирования данных на диапазон, когда пользователь располагает указатель мыши на маркере заполнения исходного диапазона и перемещает его вниз и вправо, выделяя весь диапазон, в который переносятся исходные данные.

`expression.AutoFill(Destination, Type`, где:

- *expression* — обязательный элемент, задающий диапазон, с которого начинается заполнение;
- *`Destination`* — обязательный параметр, определяющий диапазон, который заполняется. Этот диапазон должен содержать диапазон, указанный в *expression*;
- *`Type`* — необязательный параметр, указывающий тип заполнения. Допустимыми значениями являются следующие константы `XlAutoFillType`: `xlFillDefault`, `xlFillSeries`, `xlFillCopy`, `xlFillFormats`, `xlFillValues`, `xlFillDays`, `xlFillWeekdays`, `xlFillMonths`, `xlFillYears`, `xlLinearTrend`, `xlGrowthTrend`. По умолчанию используется тот тип заполнения, который наиболее подходит к данным из диапазона, указанного в *expression*.

Например, следующие инструкции (листинг 7.59, *а*) заполняют диапазон **A1:A5** членами арифметической прогрессии, причем ее первыми двумя членами являются 1 и 3, т. е. те значения, которые предварительно были введены в ячейки **A1** и **A2** (рис. 7.9).

	A	B	C	D	E
1	1	1	Лето 2000	Январь	Январь
2	3	3	Лето 2001	Февраль	Январь
3	5	9	Лето 2002	Март	Январь
4	7	27			
5	9	81			

Рис. 7.9. Последовательности

Листинг 7.59, *а*. Последовательности

```
Range("A1").Value = 1
Range("A2").Value = 3
Range("A1:A2").AutoFill Destination:=Range("A1:A5"), Type:=xlLinearTrend
```

Следующие инструкции (листинг 7.59, *б*) выводят члены геометрической прогрессии с теми же двумя начальными значениями в диапазон **B1:B5** (см. рис. 7.9).

Листинг 7.59, *б*. Последовательности

```
Range("B1").Value = 1
Range("B2").Value = 3
Range("B1:B2").AutoFill Destination:=Range("B1:B5"), Type:=xlGrowthTrend
```

Следующие инструкции (листинг 7.59, *в*) выведут в диапазон **C1:C3** последовательность значений `Лето 2000`, `Лето 2001` и `Лето 2002` с шагом 1, определенным по умолчанию методом `AutoFill` (см. рис. 7.9).

Листинг 7.59, *в*. Последовательности

```
Range("C1").Value = "Лето 2000"
Range("C1").AutoFill Destination:=Range("C1:C3"), Type:=xlFillSeries
```

Следующие две инструкции (листинг 7.59, *г*) выводят в диапазон **D1:D3** первые три члена списка — имена месяцев, начиная с `Январь` (см. рис. 7.9).

Листинг 7.59, *г*. Последовательности

```
Range("D1").Value = "Январь"
Range("D1").AutoFill Destination:=Range("D1:D3"), Type:=xlFillSeries
```

А следующие инструкции (листинг 7.59, *д*) перекопируют содержание ячейки **E1** на все ячейки диапазона **E1:E3** (см. рис. 7.9).

Листинг 7.59, *д*. Последовательности

```
Range("E1").Value = "Январь"
Range("E1").AutoFill Destination:=Range("E1:E3"), Type:=xlCopy
```

Табуляция функции

Метод `AutoFill` позволяет решить задачу табуляции функции, т. е. вывода ее значений при изменении значения ее параметра. Например, требуется найти значения функции Sin(*x*) при значении параметра *x*, изменяющегося от 0 до 2 с шагом 0,2. Это можно сделать следующим образом (листинг 7.60). Сначала в ячейку **A1** ввести первый член арифметической последовательности требуемых значений параметра, а затем с помощью метода `DataSeries` построить и всю последовательность вдоль столбца **A**. Затем определить, как текущий, диапазон с этими значениями. Диапазон, в котором расположатся соответствующие значению функции, разместится в столбце **B**, и его можно найти при помощи свойства `Offset`. Далее остается самая малость. В ячейку **B1** ввести формулу `=SIN(A1)` для нахождения значения функции при значении параметра, равным 0, а затем скопировать данную формулу на весь диапазон, отводимый под искомые значения функции (рис. 7.10).

	A	B
1	0,0	0,0000
2	0,2	0,1987
3	0,4	0,3894
4	0,6	0,5646
5	0,8	0,7174
6	1,0	0,8415
7	1,2	0,9320
8	1,4	0,9854
9	1,6	0,9996
10	1,8	0,9738
11	2,0	0,9093

Рис. 7.10. Табуляция функции

Листинг 7.60. Табуляция функции

```
Sub DemoDataSeries()
   Range("A1").Value = 0
```

```
    Range("A1").DataSeries Rowcol:=xlColumns, _
                           Type:=xlDataSeriesLinear, _
                           Step:=0.2, _
                           Stop:=2
    Dim rgn As Range
    Set rgn = Range("A1").CurrentRegion
    rgn.NumberFormat = "0.0"
    Set rgn = rgn.Offset(0, 1)
    Range("B1").Formula = "=SIN(A1)"
    Range("B1").AutoFill Destination:=rgn, Type:=xlCopy
    rgn.NumberFormat = "0.0000"
End Sub
```

Заполнение диапазона по одному значению

Метод `FillDown` объекта `Range` заполняет сверху вниз диапазон на основе значений, расположенных в верхней строчке этого диапазона, причем данные значения скопируются во все остальные ячейки диапазона.

Метод `FillUp` объекта `Range` заполняет снизу вверх диапазон на основе значений, расположенных в нижней строчке этого диапазона.

Метод `FillLeft` объекта `Range` заполняет справа налево диапазон на основе значений, расположенных в крайнем правом столбце этого диапазона.

Метод `FillRight` объекта `Range` заполняет слева направо диапазон на основе значений, расположенных в крайнем левом столбце этого диапазона.

Например, в данной инструкции содержание ячейки **A10** копируется в каждую из ячеек диапазона **A1:A9**.

```
Range("A1:A10").FillUp
```

Очистка ячейки

Методы `Clear`, `ClearComments`, `ClearContents`, `ClearFormats` и `ClearNotes` объекта `Range` очищают диапазон и в диапазоне комментарии, содержание, форматы и примечания. Например, следующая инструкция очищает диапазон **A1:G37**.

```
Range("A1:G37").Clear
```

Копирование, вырезание и удаление данных из диапазона

Метод `Copy` объекта `Range` копирует диапазон в другой диапазон или в буфер обмена.

```
Copy(Destination)
```

Здесь *Destination* — необязательный параметр, определяющий диапазон, в который копируется данный диапазон. Если этот параметр опущен, то копирование происходит в буфер обмена. В данном примере диапазон **A1:D4** активного рабочего листа копируется в диапазон **E5:H8** рабочего листа **Лист2**.

```
Range("A1:D4").Copy Worksheets("Лист2").Range("E5")
```

Метод Cut объекта Range вырезает, т. е. копирует с удалением диапазон в указанный диапазон или в буфер обмена.

```
Cut(Destination)
```

Здесь *Destination* — необязательный параметр, задающий диапазон, в который копируется данный диапазон. Если этот параметр опущен, то диапазон копируется в буфер обмена. В данном примере диапазон **A1:D4** рабочего листа **Лист1** копируется с удалением в буфер обмена.

```
Worksheets("Лист1").Range("A1:D4").Cut
```

Метод Delete объекта Range удаляет диапазон. В данном примере удаляется третья строка активной рабочей страницы.

```
Rows(3).Delete
```

Специальная вставка

Метод PasteSpecial объекта Range производит специальную вставку (*special paste*) из буфера обмена. Этот метод программирует выполнение на рабочем листе команды **Правка | Специальная вставка**.

expression.PasteSpecial(*Paste*, *Operation*, *SkipBlanks*, *Transpose*), где:

- *expression* — объект типа Range, задающий диапазон или ссылку на левую верхнюю ячейку диапазона, в который вставляются данные из буфера обмена;
- *Paste* — необязательный параметр. Определяет ту часть содержания буфера обмена, которая должна быть вставлена в диапазон. Допустимыми значениями являются следующие константы XlPasteType: xlPasteAll, xlPasteAllExceptBorders, xlPasteColumnWidths, xlPasteComments, xlPasteFormats, xlPasteFormulas, xlPasteFormulasAndNumberFormats, xlPasteValidation, xlPasteValues и xlPasteValuesAndNumberFormats;
- *Operation* — необязательный параметр. Определяет операцию над вставляемыми данными и теми, которые содержатся в диапазоне. Допустимыми значениями являются следующие константы XlPasteSpecialOperation: xlPasteSpecialOperationAdd, xlPasteSpecialOperationDivide, xlPasteSpecialOperationMultiply, xlPasteSpecialOperationNone и xlPasteSpecialOperationSubtract;
- *SkipBlanks* — необязательный параметр. Принимает логические значения и устанавливает, учитываются ли пустые ячейки при вставке;

- *Transpose* — необязательный параметр. Принимает логические значения и устанавливает, вставляется ли диапазон транспонированным.

В приведенном ниже коде данные из диапазона **C1:C5** рабочего листа **Лист1** вставляются в диапазон **D1:D5** того же листа, причем они не заменяют уже существующие данные, а прибавляют к ним данные из диапазона **C1:C5**.

```
With Worksheets("Лист1")
    .Range("C1:C5").Copy
    .Range("D1:D5").PasteSpecial Operation:=xlPasteSpecialOperationAdd
End With
```

Тот же результат можно получить при помощи следующих инструкций, где метод применяется не ко всему диапазону, а только к его левой верхней ячейке.

```
With Worksheets("Лист1")
    .Range("C1:C5").Copy
    .Range("D1").PasteSpecial Operation:=xlPasteSpecialOperationAdd
End With
```

Следующий код копирует данные из диапазона **A1:C1** в буфер обмена и затем вставляет только значения в диапазон **A5:C5**.

```
Range("A1:C1").Copy
Range("A5").PasteSpecial Paste:=xlPasteValues, Operation:=xlNone
```

Вставка диапазона с транспонированием

Вставка диапазона с транспонированием осуществляется методом `PasteSpecial` при значении его параметра `Transpose` равным `True`. Например, в следующем коде (листинг 7.61) значения из диапазона **A1:C2** копируются с транспонированием в диапазон **D1:E3**.

Листинг 7.61. Вставка диапазона с транспонированием

```
Range("A1:C2").Copy
Range("D1).PasteSpecial Paste:=xlPasteAll, _
                        Operation:=xlNone, _
                        Transpose:=True
```

Копирование диапазона в буфер обмена как растровое изображение

Метод `CopyPicture` объекта `Range` копирует в буфер обмена как растровое изображение содержимое указанного диапазона.

CopyPicture(*Appearance*, *Format*), где:

- *Appearance* — необязательный параметр, определяющий способ копирования диапазона. Допустимыми значениями являются следующие константы XlPictureAppearance: xlPrinter и xlScreen;
- *Format* — необязательный параметр, задающий формат изображения. Допустимыми значениями являются следующие константы XlCopyPictureFormat: xlPrinter и xlPicture.

Например, в следующем коде (листинг 7.62) диапазон **A1:C2** копируется в буфер обмена как растровое изображение, затем он вставляется в лист, но уже как графический объект, левый верхний угол которого совпадает с соответствующим углом ячейки **C7**, после чего это изображение поворачивается на 45° против часовой стрелки.

Листинг 7.62. Копирование диапазона в буфер обмена как растровое изображение

```
Range("A1:C2").CopyPicture
Range("C7").Select
ActiveSheet.Paste
Selection.ShapeRange.IncrementRotation -45
```

Бегущая картинка

В качестве примера копирования в буфер обмена изображения диапазона приведем следующий код (листинг 7.63). В нем в диапазон **A1** вставляется текст, у которого задается стиль шрифта. Затем он поворачивается на угол 45 градусов и содержимое ячейки копируется в буфер обмена, после чего снова вставляется в документ, но уже как рисунок. После чего рисунок начинается передвигаться по рабочему листу.

Листинг 7.63. Бегущая картинка

```
Sub RunningPicture()
   Dim txt As String, angle As Single, st As Single
   Dim pic As Object
   txt = "Hello, World!"
   angle = 45
   With Application
      .ScreenUpdating = False
      With Range("A1")
         .Value = txt
```

```
            .Font.Bold = True
            .Font.Color = RGB(233, 113, 229)
            .Font.Size = 16
            .Orientation = angle
            .EntireColumn.AutoFit
            .Copy
            Set pic = ActiveSheet.Pictures.Paste(Link:=False)
            .Clear
            With pic
               .Top = 0
               .Left = 0
            End With
         End With
         .CutCopyMode = False
         .ScreenUpdating = True
         MsgBox "Start!"
         With pic
            For st = 0 To 100
               .Top = st
               .Left = st
            Next
            .Delete
          End With
      End With
      Set pic = Nothing
End Sub
```

Поиск значения в диапазоне

Метод `Find` объекта `Range` производит поиск специфицированной информации в указанном диапазоне и возвращает ссылку на первую ячейку, в которой требуемая информация найдена. В случае необнаружения искомых данных метод возвращает значение `Nothing`.

`Find(`*`What`*`, `*`After`*`, `*`LookIn`*`, `*`LookAt`*`, `*`SearchOrder`*`, `*`SearchDirection`*`, ↳`*`MatchCase`*`, `*`MatchByte`*`, `*`SearchFormat`*`)`, где:

- *`What`* — обязательный параметр, задающий искомые данные;
- *`After`* — необязательный параметр, указывающий на ячейку, после которой надо производить поиск;
- *`LookIn`* — необязательный параметр, специфицирующий, где производить поиск. Допустимыми значениями являются следующие константы `XlFindLookIn`: `xlComments`, `xlFormulas` и `xlValues`;

- *LookAt* — необязательный параметр, указывающий, как надо искать. Допустимыми значениями являются следующие константы XlLookAt: xlWhole и xlPart;
- *SearchOrder* — необязательный параметр, задающий порядок просмотра диапазона. Допустимыми значениями являются следующие константы XlSearchOrder: xlByRows и xlByColumns;
- *SearchDirection* — необязательный параметр, определяющий направление просмотра. Допустимыми значениями являются следующие константы XlSearchDirection: xlNext и xlPrevious;
- *MatchCase* — необязательный параметр, указывающий, надо ли при поиске учитывать регистр букв;
- *MatchByte* — необязательный параметр, применяется редко;
- *SearchFormat* — необязательный параметр, задающий формат поиска.

Например, следующий код (листинг 7.64) производит поиск значения 17 в диапазоне **A1:A10**. В случае его обнаружения на экране отображается окно с адресом обнаруженной ячейки.

Листинг 7.64. Поиск значения

```
Dim rng As Range
Set rng = Range("A1:A10").Find(What:=17, LookIn:=xlValues)
If Not (rng Is Nothing) Then
   MsgBox rng.Address
Else
   MsgBox "Не найдено значение"
End If
```

Следующий код (листинг 7.65) производит поиск подстроки BHV без учета регистра в диапазоне **A1:A10**. В случае успешного поиска на экране отображается окно со значением свойства Value обнаруженной ячейки.

Листинг 7.65. Поиск подстроки без учета регистра

```
Sub DemoFindNoMatchCase ()
   Dim rng As Range
   Set rng = Range("A1:A10").Find(What:="BHV", LookIn:=xlValues, _
                                  LookAt:=xlPart, MatchCase:=False)
   If Not (rng Is Nothing) Then
      MsgBox rng.Value
   Else
```

```
      MsgBox "Не найдено подходящее значение"
   End If
End Sub
```

Повторный поиск и поиск всех значений

Метод `FindNext` и `FindPrevious` объекта `Range` реализует повторный вызов метода `Find` для продолжения специфицированного поиска. Первый из методов производит поиск следующей ячейки, а второй — поиск предыдущей, удовлетворяющей объявленным критериям поиска.

```
FindNext(After)
FindPrevious(After)
```

Здесь *After* — необязательный параметр, указывающий на ячейку, после которой надо производить поиск.

В качестве примера приведем код (листинг 7.66), который производит поиск подстроки `BHV` без учета регистра в диапазоне **A1:A10**. Все найденные ячейки заливаются желтым цветом.

Листинг 7.66. Нахождение всех вхождений подстроки в данный диапазон

```
Sub DemoFind()
   Dim firstAddress As String
   Dim rng As Range
   Set rng = Range("A1:A10").Find(What:="MS", LookIn:=xlValues, _
                                  LookAt:=xlPart, MatchCase:=False)
   If Not (rng Is Nothing) Then
      firstAddress = rng.Address
      Do
         rng.Interior.Color = RGB(255, 255, 0)
         Set rng = Range("a1:a10").FindNext(rng)
      Loop While Not (rng Is Nothing) And rng.Address <> firstAddress
   End If
End Sub
```

Замена значений

Метод `Replace` объекта `Range` производит замену в указаном диапазоне.

`Replace(`*What*`, `*Replacement*`, `*LookAt*`, `*SearchOrder*`, `*MatchCase*`, `*MatchByte*`, `*SearchFormat*`, `*ReplaceFormat*`)`, где:

- *What* — обязательный параметр, задающий строку, которая будет заменяться;

- *Replacement* — обязательный параметр, задающий строку, которой будет производиться замещение;
- *LookAt* — необязательный параметр, указывающий, как надо искать. Допустимыми значениями являются следующие константы XlLookAt: xlWhole и xlPart;
- *SearchOrder* — необязательный параметр, задающий порядок просмотра диапазона. Допустимыми значениями являются следующие константы XlSearchOrder: xlByRows и xlByColumns;
- *SearchDirection* — необязательный параметр, определяющий направление просмотра. Допустимыми значениями являются следующие константы XlSearchDirection: xlNext и xlPrevious;
- *MatchCase* — необязательный параметр, указывающий, надо ли при поиске учитывать регистр букв;
- *MatchByte* — необязательный параметр, применяется редко;
- *SearchFormat* — необязательный параметр, задающий формат поиска;
- *ReplaceFormat* — необязательный параметр, задающий формат замены.

Например, в следующем коде в столбце **A** строка MS заменяется строкой Microsoft.

```
Columns("A").Replace What:="MS", Replacement:="Microsoft", _
                     SearchOrder:=xlByColumns, MatchCase:=True
```

Подбор параметра и решение уравеняя с одной неизвестной

Метод GoalSeek объекта Range подбирает значение параметра (неизвестной величины), являющееся решением уравнения с одной переменной. Предполагается, что уравнение приведено к следующему виду: правая часть уравнения является постоянной, не зависящей от параметра, параметр входит только в левую часть уравнения, например,

$$x^3 - 3x - 5 = 0$$

Метод GoalSeek программирует выполнение команды **Сервис | Подбор параметра**. Этот метод вычисляет корень, используя метод последовательных приближений, результат выполнения которого, вообще говоря, зависит от начального приближения. Поэтому для корректности нахождения корня надо позаботиться о корректном указании этого начального приближения.

expression.GoalSeek(*Goal*, *ChangingCell*), где:

- *expression* — ячейка, в которую введена формула, являющаяся правой частью решаемого уравнения. В этой формуле роль параметра (неизвестной величины) играет ссылка на ячейку, указанную в аргументе ChangingCell;

- *Goal* — обязательный параметр, задающий значение левой части решаемого уравнения, не содержащей параметра;
- *ChangingCell* — обязательный параметр, указывающий ссылку на ячейку, отведенную под параметр (неизвестную величину). Значение, введенное в данную ячейку до активизации метода GoalSeek, рассматривается как начальное приближение к искомому корню. Значение, возвращаемое в эту ячейку после выполнения метода GoalSeek, является найденным приближением к корню.

Точность, с которой находится корень и предельно допустимое число итераций, используемых для нахождения корня, устанавливается свойствами MaxChange и MaxIterations объекта Application. Например, определение корня с точностью до 0,0001 максимум за 1000 итераций устанавливается инструкцией:

```
With Application
   .MaxIterations = 1000
   .MaxChange = 0.0001
End With
```

Метод GoalSeek возвращает значение True, если решение найдено, и значение False в противном случае.

Например, следующий код (листинг 7.67) ищет корень уравнения $x^3 - 3x - 5 = 0$ при начальном приближении 1 (рис. 7.11).

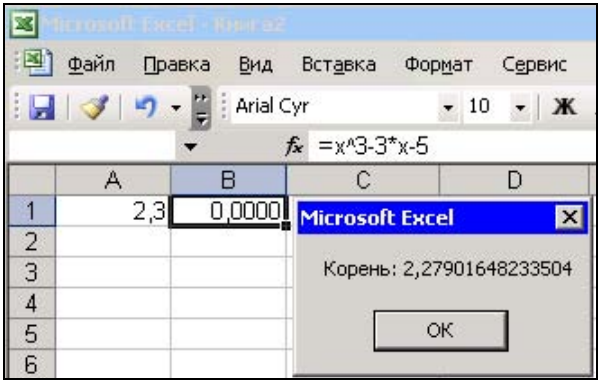


Рис. 7.11. Решение уравнения

Листинг 7.67. Решение уравнения

```
Sub DemoGoalSeek()
   Range("A1").Name = "x"
   Range("A1").Value = 1
```

```
   Range("B1").Formula = "=x^3-3*x-5"
   If Range("B1").GoalSeek(Goal:=0, ChangingCell:=Range("x")) Then
      MsgBox "Корень: " & Range("A1").Value
   Else
      MsgBox "Корень не найден"
   End If
End Sub
```

Ввод в диапазон неповторяющихся значений

Рассмотрим бизнес-ситуацию, в которой в диапазон рабочего листа надо ввести список сотрудников с различными фамилиями, причем ввод автоматически производить в алфавитном порядке (рис. 7.12). Итак:

- ❒ введите в ячейку **A1** значение **Фамилия**;
- ❒ введите в ячейку **B1** значение **Новый сотрудник;**
- ❒ в диапазон ячеек, начиная с ячейки **A2,** вниз вдоль столбца **A** будут вводиться фамилии отобранных сотрудников;
- ❒ в ячейку **C1** будет вводиться имя очередного сотрудника;
- ❒ на рабочем листе расположите кнопку. При помощи окна **Properties** установите у нее значения свойств, как показано в табл. 7.25;

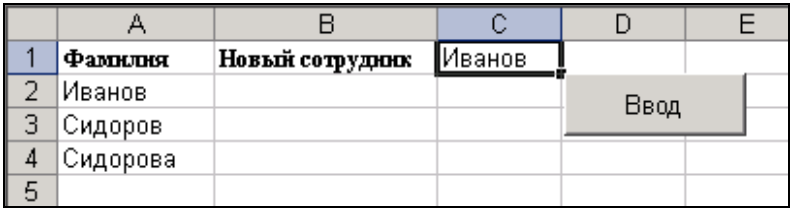


Рис. 7.12. Ввод в диапазон неповторяющихся значений

Таблица 7.25. *Значения свойств, установленных в окне* ***Properties***

Свойство	Значение
`Name`	`cmdEnter`
`Text`	`Ввод`

- ❒ в модуле рабочего листа наберите следующий код (листинг 7.68). Поиск подходящего значения производится методом `Match` объекта `Application`.

Листинг 7.68. Ввод в диапазон неповторяющихся значений

```
Private Sub cmdEnter_Click()
   Application.DisplayAlerts = False
   If IsError(Application.Match(Range("C1"), _
              Range(Range("A2"), Range("A2").End(xlDown)), 0)) Then
      If Range("A1").CurrentRegion.Rows.Count > 1 Then
         With Range("A1").End(xlDown).Offset(1, 0)
            .Value = Range("C1").Value
            .Sort Key1:=Range("A2"), Order1:=xlAscending, Header:=xlYes
         End With
      Else
         Range("A2").Value = Range("C1").Value
      End If
   Else
      MsgBox ("Уже существует")
   End IfEnd Sub
```

Отсылка электронной почты

Отсылка электронной почты с данными из рабочего листа может производиться при помощи средств Microsoft Outlook. Продемонстрируем, как это можно сделать на примере, в котором при нажатии кнопки из ячеек рабочего листа считывается адрес получателя, само сообщение, его тема и адрес получателя, которому будет отослана копия сообщения. Для этого сделайте следующие действия:

- установите ссылку на библиотеку объектов Microsoft Outlook. С этой целью выберите команду **Tools | References**. На экране отобразится окно **References**. В списке **Available References** установите флажок **Microsoft Outlook 11.0 Object Library** и нажмите кнопку **OK**;
- на рабочем листе введите в ячейку **B1** электронный адрес получателя, в ячейку **B2** — текст отправляемого сообщения, в ячейку **B3** — тему сообщения, в ячейку **B4** — адрес получателя копии сообщения (рис. 7.13);

	A	B
1	Адрес получателя	winnie@rambler.ru
2	Сообщение	Как дела?
3	Тема сообщения	Тест
4	Адрес получателя копии сообщения	james@hotbox.ru
5		
6		Послать почту

Рис. 7.13. Отсылка электронной почты

- создайте кнопку. Используя окно **Properties**, установите у нее значения свойств `Name` и `Caption` равными `cmdEMail` и `Послать почту`. На рабочем листе наберите код из листинга 7.69. Проект готов.

Листинг 7.69. Отсылка электронной почты

```
Private Sub cmdEMail_Click()
   Dim objOL As New Outlook.Application
   Dim objMail As MailItem
   Set objOL = New Outlook.Application
   Set objMail = objOL.CreateItem(olMailItem)
   With objMail
      .To = Range("B1").Value
      .Body = Range("B2").Value
      .Subject = Range("B3").Value
      .CC = Range("B4").Value
      .Display
   End With
   Set objMail = Nothing
   Set objOL = Nothing
End Sub
```

Озвучивание текста

Объект `Speech` инкапсулирует в себе средства по озвучиванию текста, содержащегося в ячейках рабочего листа. Метод `Speak` реализует непосредственно произношение, свойство `Direction` задает порядок прочитывания текста (допустимые значения: `xlSpeakByColumns` и `xlSpeakByRows`), а свойство `SpeakCellOnEnter` устанавливает или отменяет режим чтения текста по его вводу в ячейку. В следующем коде (листинг 7.70) процедура `Speaker` читает текст из ячеек указанного диапазона, а процедура `SpeakerString` читает указанную строку текста.

Листинг 7.70. Озвучивание текста

```
Sub TestSpeaker()
   Range("C2").Value = "Hello, everybody!"
   Range("D2").Value = "How are you?"
   Speaker Range("C2:D2"), xlSpeakByRows
   SpeakerString "Not too bad"
```

```
End Sub

Sub Speaker(r As Range, direction As Integer)
   Application.Speech.direction = direction
   r.Speak
End Sub

Sub SpeakerString(s As String)
   Application.Speech.Speak s
End Sub
```

Условное форматирование

Условное форматирование позволяет эффективно отображать, форматируя ячейки выборочно, основываясь на их содержании. Для того чтобы применить условное форматирование к ячейке или диапазону:

- выделите ячейку или диапазон, например **B2:B5**;
- выберите команду **Формат | Условное форматирование**;
- отобразится окно **Условное форматирование**. Для простого условного форматирования, как в данном случае, из раскрывающегося списка выберите **значение**. Если же необходимо форматирование на основе формулы, то надо выбрать **формула**;
- определите условие;
- нажмите кнопку **Формат** и определите форматирование (шрифта, границы и заливки), которое надо применить;
- для добавления дополнительных условий нажмите кнопку **А также**;
- нажмите кнопку **ОК**.

Примечание

При условном форматировании нельзя ссылаться на ячейки, расположенные на других листах. Для того чтобы избежать это ограничение, на исходном листе создайте формулу, ссылающуюся на ячейку другого листа. Например, если необходимо в формуле сослаться на ячейку **A1** листа **Лист2**, то на исходном листе в какую-либо ячейку введите формулу `=Лист2!A1`, а при условном форматировании ссылку давайте уже на эту ячейку. Допустимо также присвоение имени диапазону, после чего на него разрешена ссылка при условном форматировании на любом рабочем листе.

Для удаления условного форматирования совместно со всем другими форматами выделите искомый диапазон и выберите команду **Правка | Очистить | Формат**.

Для удаления только условного форматирования выделите искомый диапазон, выберите команду **Формат | Условное форматирование** и в окне **Условное форматирование** нажмите кнопку **Удалить**. Отобразится окно **Удаление условного форматирования,** предлагающее указать, какие из трех условий следует удалить, даже если такое количество условий и не было определено.

В качестве примера создадим приложение, в котором к таблице данных применяются три различных типа условного форматирования (рис. 7.14). Итак, в диапазон **B1:B6** введена некоторая отчетная таблица. Кроме того, имеются три переключателя: **Не меньше среднего**, **Максисум** и **Не меньше указанного значения**, а в ячейку **G8** введен список значений на основе данных из диапазона **G1:G3**.

- Выбор переключателя **Не меньше среднего** вызывает окрашивание в желтый цвет всех ячеек из диапазона **B1:B6,** значения которых не меньше среднего.
- Выбор переключателя **Максисум** приводит к выделению всех значений, совпадающих с максимальным полужирным красным шрифтом.
- Выбор переключателя **Не меньше указанного значения** вызывает окрашивание в зеленый цвет всех ячеек из диапазона **B1:B6,** значения которых не меньше, чем значение из ячейки **G8**.

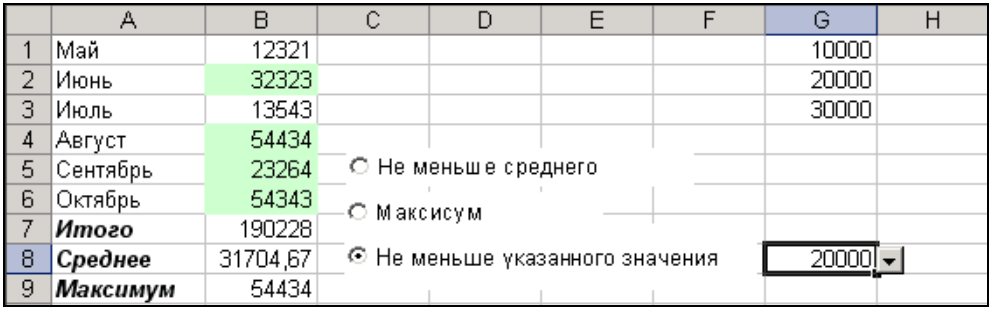


Рис. 7.14. Условное форматирование

Итак, перейдем к созданию проекта:

- на рабочем листе расположите три переключателя. При помощи окна **Properties** установите у них значения свойств, как показано в табл. 7.26;

Таблица 7.26. *Значения свойств элементов управлений, установленных в окне* ***Properties***

Элемент управления	Свойство	Значение
Переключатель	`Name`	`optAverage`
	`Caption`	`Не меньше среднего`

Таблица 7.26 *(окончание)*

Элемент управления	Свойство	Значение
Переключатель	Name	optMax
	Caption	Максисум
Переключатель	Name	optValue
	Caption	Не меньше указанного значения

- в модуле рабочего листа введите код из листинга 7.71;
- в ячейку **B7** введите формулу
  `=СУММ(B1:B6);`
- в ячейку **B8** введите формулу
  `=СРЗНАЧ(B1:B6);`
- в ячейку **B9** введите формулу
  `=МАКС(B1:B6);`
- в ячейку **G8** введите список значений на основе данных из диапазона **G1:G3**.

Листинг 7.71. Условное форматирование

```
Private Sub optAverage_Click()
   Dim r As Range
   Set r = Range("B1:B6")
   r.FormatConditions.Delete
   r.FormatConditions.Add Type:=xlExpression, _
                          Formula1:="=B1>=СРЗНАЧ($B$1:$B$6)"
   r.FormatConditions(1).Interior.Color = RGB(255, 255, 0)
End Sub

Private Sub optMax_Click()
   Dim r As Range
   Set r = Range("B1:B6")
   r.FormatConditions.Delete
   r.FormatConditions.Add Type:=xlCellValue, _
                          Operator:=xlEqual, _
                          Formula1:="=$B$9"
   With r.FormatConditions(1).Font
      .Bold = True
```

```
      .Italic = False
      .Color = RGB(255, 0, 0)
   End With
End Sub

Private Sub optValue_Click()
   Dim r As Range
   Set r = Range("B1:B6")
   r.FormatConditions.Delete
   r.FormatConditions.Add Type:=xlCellValue, _
                          Operator:=xlGreaterEqual, _
                          Formula1:="=$G$8"
   r.FormatConditions(1).Interior.Color = RGB(0, 0, 255)
End Sub
```

Данные об условном форматировании инкапсулируются в объекте `FormatCondition`, а эти объекты в своей совокупности образуют семейство `FormatConditions`. Удаление условных форматов осуществляется методом `Delete`, а создание новых — методом `Add`.

`Add(`*`Type`*`, `*`Operator`*`, `*`Formula1`*`, `*`Formula2`*`)`, где:

- *`Type`* — обязательный параметр, определяющий, построен ли формат на значении ячейки или формуле. Допустимыми значениями являются константы `XlFormatConditionType`: `xlCellValue` и `xlExpression`;
- *`Operator`* — необязательный параметр, задающий оператор условного форматирования. Допустимыми значениями являются константы `XlFormatConditionOperator`: `xlBetween`, `xlEqual`, `xlGreater`, `xlGreaterEqual`, `xlLess`, `xlLessEqual`, `xlNotBetween` и `xlNotEqual`. Если значение параметра *`Type`* равно `xlExpression`, то параметр *`Operator`* игнорируется;
- *`Formula1`* — необязательный параметр, задающий либо формулу, либо выражение, либо значение, ассоциирование с оператором условного форматирования;
- *`Formula2`* — необязательный параметр, задающий либо выражение, либо значение, ассоциированное с оператором условного форматирования, когда значение параметра *`Operator`* равно `xlBetween` или `xlNotBetween`.

Обратитите внимание на то, что при условном форматировании по формуле применяется следующая конструкция, которая задает форматирование всех ячеек диапазона **B1:B6**, значения которых будут не меньше, чем среднее значение. Особо обратите внимание на то, что в этой формуле ссылка на первую ячейку **B1** диапазона **B1:B6** относительна, а в качестве значения параметра формулы `СРЗНАЧ` применяется абсолютная ссылка на диапазон.

```
=B1>=СРЗНАЧ($B$1:$B$6)
```

Примечание

Если бы надо было найти не максимальное, а, например, два наибольших значения, то надо бы было воспользоваться функцией НАИБОЛЬШИЙ, которая возвращает *n*-е наибольшее значение. В этом случае формула в условном форматировании имела бы вид:

```
=B1>=НАИБОЛЬШИЙ($B$1:$B$6)
```

Отображение результата только при условии ввода всех данных

С помощью условного форматирования можно легко обеспечить, чтобы содержимое ячеек с результатами отображалось только при вводе все данных. Например, необходимо найти сумму двух чисел, вводимых в ячейки **B1** и **B2**. Результат с пояснительными надписями будет отображаться в ячейках **C2** и **C1** только в случае, если оба слагаемых введены. Для этого достаточно диапазону **C1:C2** задать условное форматирование при помощи следующей формулы, причем установить цвет шрифта, совпадающим с фоновым цветом.

```
=СЧЁТЗ($B$1:$B$2)<>2
```

Проверка вводимых значений

Для того чтобы задать тип данных, допустимых для диапазона, надо:

- ❒ выделить диапазон;
- ❒ выбрать команду **Дата | Проверка**;
- ❒ на вкладке **Параметры** окна **Проверка вводимых значений** в списке **Тип данных** надо указать тип допустимых данных. В списке **Тип данных** имеются следующие элементы:
  - **Любое значение** — удаляет существующее правило проверки значений;
  - **Целое число** — пользователь может вводить целые числа. Диапазон допустимых значений задается при помощи раскрывающихся списков **Значение**, **Минимум** и **Максимум**. Например, можно ограничить ввод целыми числами от 1 до 100;
  - **Действительное** — пользователь может вводить действительные числа. Диапазон допустимых значений задается при помощи раскрывающихся списков **Значение**, **Минимум** и **Максимум**. Например, можно ограничить ввод действительными числами из интервала [0, 1];
  - **Список** — пользователь может выбрать значение из данного списка. Список задается в поле **Источник**, в котором указывается ссылка на диапазон с исходными данными;

- **Дата** — пользователь может вводить дату. Диапазон допустимых дат задается при помощи раскрывающихся списков **Значение**, **Минимум** и **Максимум**;
- **Время** — пользователь может вводить время. Диапазон допустимых времен задается при помощи раскрывающихся списков **Значение**, **Минимум** и **Максимум**;
- **Длина текста** — пользователь может вводить строки со специфицированными ограничениями по длине;
- **Другой** — возвращает логическое значение, определяющее, соответствует ли введенное пользователям значение ограничениям, которые задаются формулой. Ссылка на ячейку с формулой задается в поле **Формула**.

❐ на вкладке **Сообщение для ввода** установливается заголовок и текст всплывающей подсказки, отображаемой при расположении указателя мыши над ячейкой данного диапазона;

❐ на вкладке **Сообщение об ошибке** задается значок, кнопки, заголовок и сообщение окна, отображаемого при попытке ввода некоректных данных.

Наиболее интересным применением провепки типа данных является использование формулы для задания ограничений. Данный режим проверки устанавливается выбором элемента **Другой** из списка **Тип данных**. В этом случае в поле **Формула** надо ввести формулу проверки данных в относительном формате с привязкой к левой верхней ячейке данного диапазона.

Например, данные содержатся в диапазона **B2:B10**, тогда:

❐ если в ячейки этого диапазона надо вводить только строки текста, начинающиеся с буквы "А", то воспользуйтесь формулой

```
=ЛЕВСИМВ(B2)="А";
```

❐ если в ячейки этого диапазона надо вводить только строки текста, состоящие из трех символов и начинающиеся с буквы А, то воспользуйтесь формулой

```
=ЛЕВСИМВ(B2, "А??")=1;
```

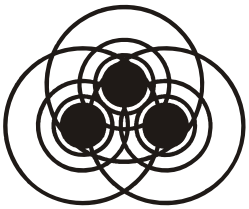
❐ если надо вводить только строки, состоящие из трех символов, то используйте формулу

```
=ДЛСТР(B2)=3;
```

❐ если можно вводить только неповторяющиеся данные, то воспользуйтесь формулой

```
=СЧЁТЕСЛИ($B$2:$B$10;B2)=1
```

Глава 8



Формы и элементы управления

При помощи форм, а также элементов управления, разработчик может создавать диалоговые окна произвольной конфигурации, максимально приспособленные для решения конкретных задач приложения. Элементы управления группируются в панели инструментов **Toolbox** редактора Visual Basic и представляют собой следующие объекты:

- Label (Надпись);
- TextBox (Поле);
- CommandButton (Кнопка);
- CheckBox (Флажок);
- OptionButton (Переключатель);
- ToggleButton (Выключатель);
- ScrollBar (Полоса прокрутки);
- SpinButton (Счетчик);
- Image (Рисунок);
- ListBox (Список);
- ComboBox (Поле со списком);
- Frame (Рамка);
- MultiPage (Вкладки);
- TabStrip (Ярлыки).

Все эти элементы можно размещать как на форме, так и на рабочем листе, придавая ему большую итеративность. Объектная модель форм включает следующие объекты и семейства:

- объект `UserForm` инкапсулирует в себе данные о самой форме;
- семейство `Controls` содержит в себе все элементы управления, перечисленные выше, расположенные на форме;
- объект `Font`, устанавливает шрифт, используемый в форме;

- объекты `MultiPage` и `TabStrip` являются контейнерами для семейств `Pages` и `Tabs`, элементы которых представляют собой отдельные страницы и вкладки соответствующих элементов управления;
- объект `DataObject` инкапсулирует в себе перемещаемые отформатированные текстовые данные.

Первая форма

Диалоговое окно является формой. Чтобы добавить форму в проект:

1. Перейдите в редактор Visual Basic.
2. Выберите команду **Insert** | **UserForm**.

Новая форма добавлена в проект (рис. 8.1).

Размеры формы можно изменять при помощи маркеров изменения размеров.

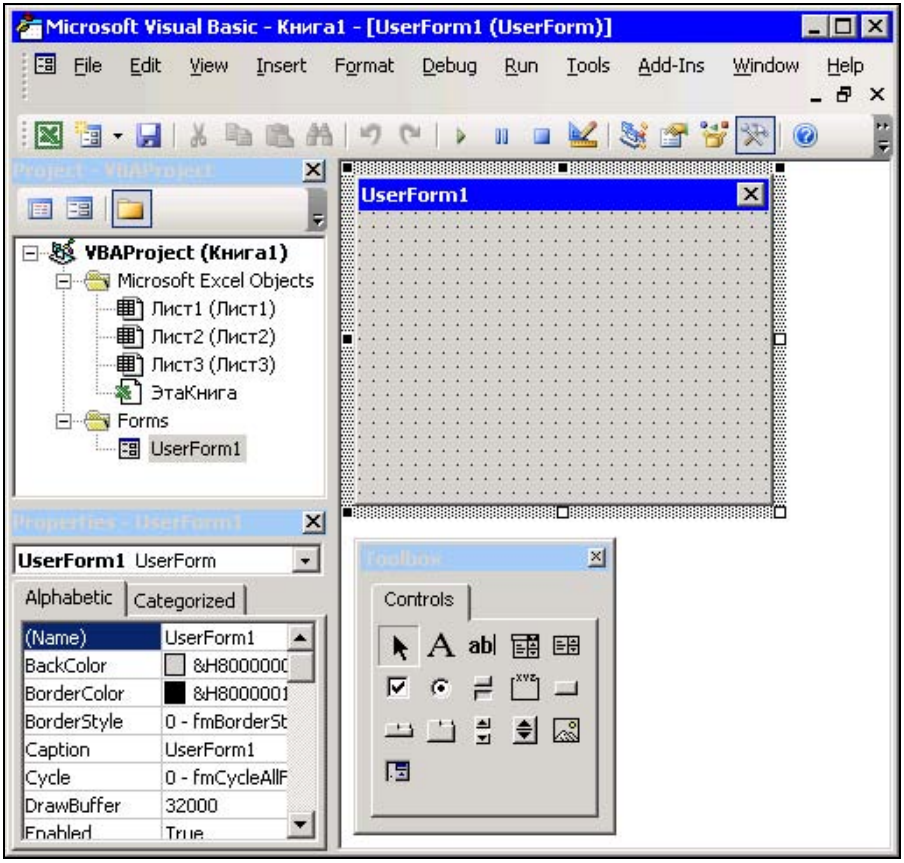


Рис. 8.1. Новая форма в редакторе Visual Basic

Форма как объект

Форма является объектом `UserForm`, обладающим большим количеством свойств, методов и событий, позволяющих контролировать ее внешний вид и функционирование. Конечно, наиболее часто используемыми свойствами формы являются те, которые задают имя формы (`Name`) и текст (`Caption`), отображаемый в заголовке окна. Свойство `Picture` задает фоновый рисунок, свойство `PictureAlignment` — его расположение, а свойство `PictureTiling` — располагается ли он в виде мозаики. Основными методами являются метод `Show`, который ее отображает и метод `PrintForm`, печатающий ее изображения. Базисными событиями являются события `Activate` и `Deactivate`, генерируемые при активизации и деактивизации формы, `QueryClose` и `Terminate`, происходящие непосредственно перед и во время закрытия формы. В качестве первого проекта с формой создадим пользовательскую форму с традиционным заголовком Hello, World!, с фоновым мозаическим рисунком, выровненным по верхнему левому углу формы, причем форма отображается на экране при нажатии кнопки, расположенной на рабочем листе (рис. 8.2).

Рис. 8.2. Первый проект с формой

- Создайте форму и, используя окно **Properties**, установите ей значения свойств, как показано в табл. 8.1.

Таблица 8.1. *Значения свойств формы, установленные в окне* ***Properties***

Свойство	Значение
Name	frmFirst
Caption	Hello, World!
Picture	Ссылка на рисунок
PictureAlignment	fmPictureAlignmentTopLeft
PictureTiling	True

- Перейдите на рабочий лист, создайте кнопку и, используя окно **Properties**, установите ей значения свойств, как показано в табл. 8.2.

Таблица 8.2. *Значения свойств кнопки, установленные в окне* ***Properties***

Объект	Свойство	Значение
Кнопка	Name	cmdShowForm
	Caption	Нажми

- В модуле рабочего листа наберите следующий код (листинг 8.1). Теперь при нажатии кнопки на экране отобразится форма.

Листинг 8.1. Первая форма. Модуль рабочего листа

```
Private Sub cmdShowForm_Click()
   frmFirst.Show
End Sub
```

Отображение формы при выборе ячейки рабочего листа

Для отображения формы нет необходимости размещать элементы управления на рабочем листе. Обрабатывая событие `SelectionChange` объекта `Worksheet`, можно сделать так, что форма отобразится при выборе ячеек из специфицированного диапазона (в данном примере из диапазона **A1:A2**), например, как это сделано в листинге 8.2.

Листинг 8.2. Отображение формы при выборе ячейки рабочего листа. Модуль рабочего листа

```
Private Sub Worksheet_SelectionChange(ByVal Target As Range)
   On Error Resume Next
   Dim r As Range
```

```
   Set r = Intersect(Range("A1:A2"), Target)
   If Not r Is Nothing Then
      frmFirst.Show
   End If
End Sub
```

Как запустить проект на исполнение

Форму можно связать с любым элементом управления, размещенным на рабочем листе. Как это делается, было показано в предыдущем примере. В последующих примерах, где основными объектами являются сама форма и функциональные возможности размещенных на ней элементов управления, мы не будем повторять то, как форма интегрируется в рабочий лист, оставляя составление соответствующего кода читателю.

Для проверки работы кода, связанного с формой, на самом деле нет необходимости создавать элементы управления на рабочем листе и с ними связывать форму. Достаточно после конструирования формы и написания кода в модуле формы выбрать команду **Run | Run Sub/UserForm**, либо нажать клавишу <F5>, либо кнопку  панели инструментов **Standard**, и форма отобразится поверх активного рабочего листа.

Ключевое слово *Me*

В коде часто используется ключевое слово `Me`, которое возвращает имя активного окна. Например, вместо кода

```
UserForm1.Caption = "Пример"
Unload UserForm1
```

где оператор `Unload` выгружает форму из памяти, обычно используют следующий код:

```
Me.Caption = "Пример"
Unload Me
```

Создание формы в коде

Форма является объектом и ее, как и любой другой объект, можно создать в коде. Следующий пример (листинг 8.3) демонстрирует то, как конструируются копии формы **UserForm1,** находящейся в проекте. Эти формы будут различаться только взаимным расположением и заголовками (рис. 8.3). В данном примере они будут отображаться в немодальном режиме. Это обеспечивается применением метода `Show` со значением необязательного параметра равным `vbModeless`. Немодальный режим позволяет создавать новые окна или работать с другими окнами без закрытия данного.

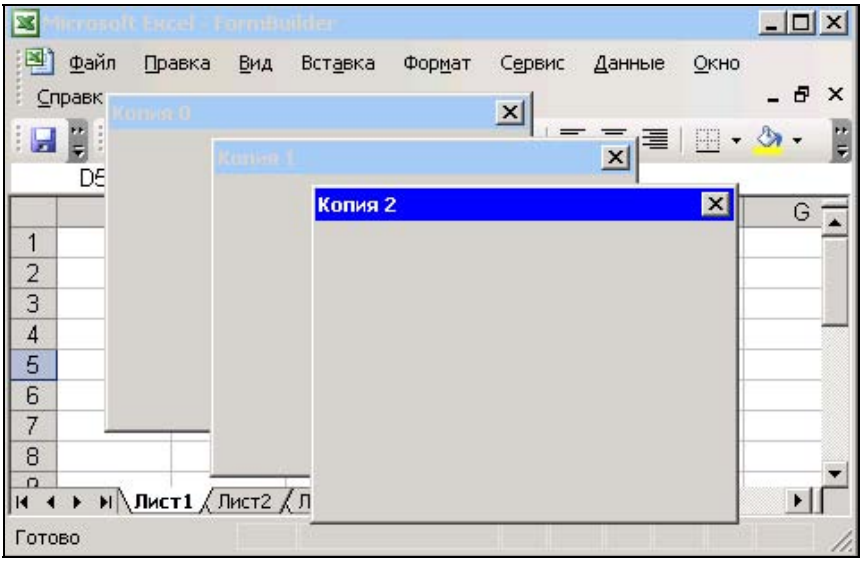


Рис. 8.3. Создание формы в коде

Листинг 8.3. Создание формы в коде

```
Sub FormsBuilder()
   Dim i As Long
   Dim frms() As UserForm1
   ReDim frms(0)
   Set frms(0) = New UserForm1
   frms(0).Caption = "Копия 0"
   frms(0).Top = 100
   frms(0).Left = 100
   frms(0).Show False
   For i = 1 To 2
      ReDim Preserve frms(i)
      Set frms(i) = New UserForm1
      frms(i).Caption = "Копия " & CStr(i)
      frms(i).Show vbModeless
      frms(i).Top = frms(i - 1).Top + 20
      frms(i).Left = frms(i - 1).Left + 30
   Next
End Sub
```

Предотвращение закрытия окна с помощью кнопки *Close*

При помощи обработки события `QueryClose`, генерируемого непосредственно перед закрытием формы, можно управлять процессом ее закрытия. У процедуры обработки этого события имеются два параметра. Если первый из них равен значению `True`, то закрытие окна блокируется. Второй параметр идентифицирует причину, вызвавшую закрытие окна, и имеет следующие допустимые значения: `vbFormControlMenu` или 0 (пользователь нажал кнопку **Close**), `vbFormCode` или 1 (в коде вызван оператор `Unload`), `vbAppWindows` или 2 (завершен текущий сеанс Microsoft Windows), `vbAppTaskManager` или 3 (Windows Task Manager закрывает приложение). Следующий код (листинг 8.4) демонстрирует то, как для пользователя можно блокировать закрытие формы при помощи кнопки **Close**.

Листинг 8.4. Предотвращение закрытия окна с помощью кнопки *Close*

```
Private Sub UserForm_QueryClose(Cancel As Integer, CloseMode As Integer)
   If CloseMode = vbFormControlMenu Then
      Cancel = True
      MsgBox "Так окно не закрыть!"
   End If
End Sub
```

Надпись

Элемент управления **Надпись (Label)** предназначен для отображения заголовков или короткого пояснительного текста. Основным свойством надписи является свойство `Caption`, задающее отображаемый текст. Свойство `BorderStyle` задает стиль границы надписи. В надписи допустим вывод небольшого растрового изображения, ссылка на которое устанавливается свойством `Picture`, а свойство `TextAlign` задает взаимное расположение рисунка и текста.

В следующем примере (листинг 8.5) на форме имеются две надписи (рис. 8.4). У первой нет границы, а в качестве сообщения выводится текущая дата. У второй надписи граница имеется, и кроме сообщения о времени старта работы приложения в нее выводится растровое изображение.

Листинг 8.5. Две надписи

```
Private Sub UserForm_Initialize()
   Label1.BorderStyle = fmBorderStyleNone
   Label1.Caption = "Сегодня: " & Format(Now(), "dd mmmm yyyy")
```

```
    Label2.BorderStyle = fmBorderStyleSingle
    Label2.Picture = LoadPicture("c:\dino.jpg")
    Label2.Caption = "Время старта программы: " _
                     & vbCr & Format(Now(), "hh:mm:ss")
    Label2.PicturePosition = fmPicturePositionLeftCenter
    Label2.TextAlign = fmTextAlignRight
End Sub
```

Рис. 8.4. Надписи

Всплывающая форма

При загрузке приложений часто первоначально на экране отображается одна или несколько **всплывающих (splash) форм**, которые несут в себе некоторую информацию о проекте. Продемонстрируем технику их создания на простом примере, имеющем две последовательно отображаемые всплывающие формы. В первой из них выводится некоторый фоновый рисунок, а во второй — сообщение в надписи. Задержка отображения формы на указанное число миллисекунд реализуется API процедурой `Sleep`. Для того чтобы приложение не зависало, во время работы этой процедуры применяется оператор `DoEvents`. Итак, создайте две формы. На второй форме расположите надпись. Используя окно **Properties**, установите ей значения свойств, как показано в табл. 8.3, а модулях наберите код из листингов 8.6 *а*, *б*, *в*, *г*.

Таблица 8.3. *Значения свойств, установленные в окне* ***Properties***

Свойство	Значение
Name	frmFirst
Caption	Hello, World!
Picture	Ссылка на рисунок
PictureAlignment	fmPictureAlignmentTopLeft
PictureTiling	True

Листинг 8.6, *а*. Всплывающая форма. Модуль `ЭтаКнига`

```
Private Sub Workbook_Open()
   UserForm1.Show
End Sub
```

Листинг 8.6, *б*. Всплывающая форма. Модуль первой формы

```
Private Sub UserForm_Activate()
   DoEvents
   Sleep 2000
   Unload UserForm1
   UserForm2.Show
End Sub
```

Листинг 8.6, *в*. Всплывающая форма. Модуль второй формы

```
Private Sub UserForm_Activate()
   DoEvents
   Sleep 2000
   Unload UserForm2
End Sub
```

Листинг 8.6, *г*. Всплывающая форма. Стандартный модуль

```
Private Declare Sub Sleep Lib "kernel32" (ByVal milliseconds As Long)
```

Отобразить и скрыть всплывающую форму можно, воспользовавшись методом `OnTime` объекта `Application`, который задает выполнение макроса

на указанное время. Например, следующий код (листинги 8.7, *а*, *б*) обеспечивает отображение формы после загрузки в течение 2 секунд.

Листинг 8.7, *а*. Всплывающая форма. Код из модуля ЭтаКнига

```
Private Sub Workbook_Open()
   Application.OnTime Now + TimeValue("00:00:2"), "HideSplash"
   UserForm1.Show
End Sub
```

Листинг 8.7, *б*. Всплывающая форма. Код из стандартного модуля

```
Sub HideSplash()
   Unload UserForm1
End Sub
```

Кнопка

Элемент управления **Кнопка (CommandButton)** предназначен для инициализации, окончания или прерывания каких-либо действий. Например, кнопка

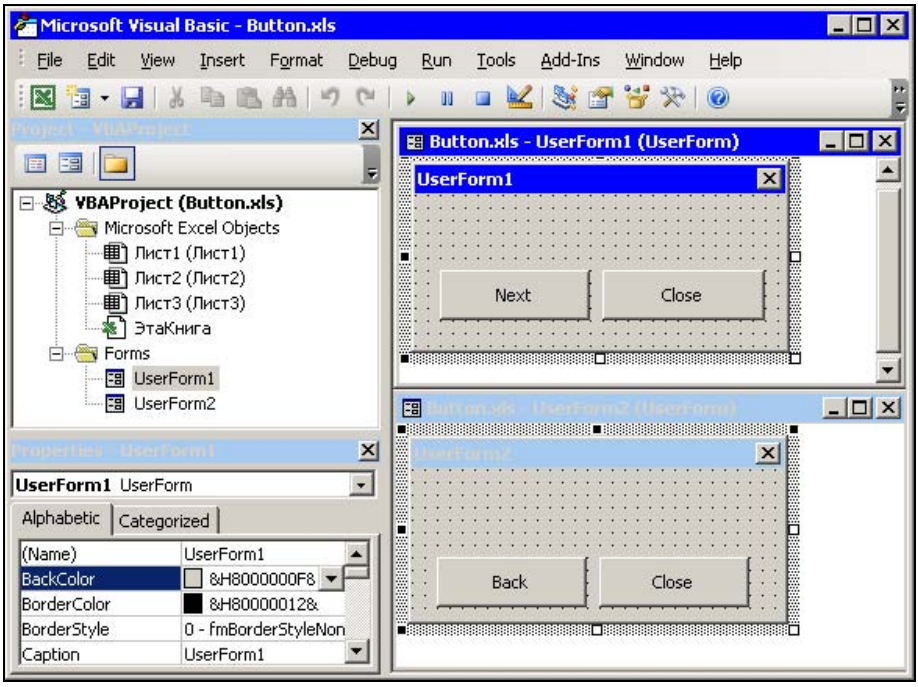


Рис. 8.5. Первая и вторая формы

может быть использования для закрытия формы, или загрузки другой формы. Основным свойством кнопки является свойство `Caption`, задающее отображаемый на ней текст. Основным событием является событие `Click`, генерируемое при ее нажатии.

В следующем примере в проекте имеются две формы. На каждой из них расположено по две кнопки. На первой — кнопки **Next** и **Close**, на второй — **Back** и **Close** (рис. 8.5). Нажатие кнопок **Next** и **Back** приводит к закрытию активной формы и отображению неактивной. Нажатие кнопки **Close** останавливает работу приложения.

Итак, создайте две формы. Используя окно **Properties**, установите им значения свойств, как показано в табл. 8.4, а в модулях форм наберите код из листингов 8.8, *а*, *б*.

Таблица 8.4. *Значения свойств элементов управления, установленные в окне* ***Properties***

Элемент управления	Свойство	Значение
Кнопка	`Name`	`cmdNext`
	`Caption`	`Next`
Кнопка	`Name`	`cmdClose`
	`Caption`	`Close`
Кнопка	`Name`	`cmdBack`
	`Caption`	`Back`
Кнопка	`Name`	`cmdClose`
	`Caption`	`Close`

Листинг 8.8, *а*. Отображение и закрытие форм. Модуль первой формы

```
Private Sub cmdNext_Click()
   Unload Me
   UserForm2.Show
End Sub

Private Sub cmdClose_Click()
   End
End Sub
```

Листинг 8.8, *б*. Отображение и закрытие форм. Модуль второй формы

```
Private Sub cmdBack_Click()
   Unload Me
   UserForm1.Show
End Sub

Private Sub cmdClose_Click()
   End
End Sub
```

Событие *Click*

Если у кнопки, расположенной на рабочем листе, надо обработать только событие `Click`, то кнопку можно заменить любым другим объектом, например, рисунком или WordArt объектом, а затем в контекстном меню этого объекта выбрать команду **Назначить макрос** и назначить объекту соответствующий макрос.

Поле

Элемент управления **Поле (TextBox)** предназначен для отображения информации, вводимой пользователем, данных, получаемых в результате различных запросов к рабочим листам, базам данных, а также отображения результатов вычислений. Если поле связано с источником данных, то изменение содержания поля ввода вызывает и изменение данных в источнике. Основным свойством поля является свойство `Text`, задающее отображаемые в нем данные. В следующем примере создается простейшее приложение по вводу данных (фамилия и имя) о клиенте в таблицу рабочего листа (рис. 8.6).

Создайте форму, на которой расположите две надписи, два поля и кнопку. Используя окно **Properties**, установите им значения свойств, как показано в табл. 8.5

Таблица 8.5. *Значения свойств элементов управления, установленные в окне* ***Properties***

Элемент управления	Свойство	Значение
Надпись	`Caption`	`Фамилия`
Надпись	`Caption`	`Имя`
Поле	`Name`	`txtLastName`
Поле	`Name`	`txtFirstName`
Кнопка	`Name`	`cmdOK`
	`Caption`	`OK`

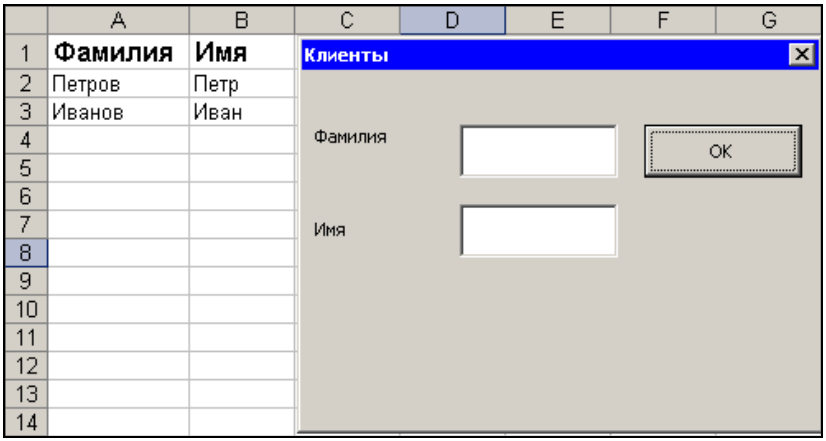


Рис. 8.6. Ввод значений из формы в рабочий лист

В ячейки **A1** и **B1** рабочего листа введите `Фамилия` и `Имя`, а в модуль формы — следующий код (листинг 8.9).

Листинг 8.9. Ввод значений из формы в рабочий лист

```
Private Sub cmdOK_Click()
   Dim n As Integer
   n = Range("A1").CurrentRegion.Rows.Count
   Cells(n + 1, 1).Value = txtLastName.Text
   Cells(n + 1, 2).Value = txtFirstName.Text
End Sub
```

Список

Элемент управления **Список (ListBox)** предназначен для отображения списка значений и позволяет пользователю выбрать одно из них. Методы `AddItem`, `RemoveItem` и `Clear` добавляют, удаляют элемент списка и полностью его очищают. Свойства `ListCount`, `Text` и `ListIndex` возвращают число элементов списка, выбранный элемент и его индекс. Свойство `MultiSelect` позволяет установить режим выбора нескольких элементов из списка, тогда свойство `Selected` определяет, выбран ли элемент со специфицированным индексом. Свойства `ColumnCount` и `ColumnWidths` устанавливают количество столбцов списка и их ширину. Свойство `List` задает ссылку на элемент, стоящий на пересечении указанных строк и столбцов. Кроме того, свойство `List`, как и свойство `Column` задает весь массив элементов списка. Основным событием списка является `Change`, генерируемое при смене выбранного элемента.

Следующий пример демонстрирует работу со списком (листинг 8.10). На форме расположен список, который заполнен названиями цветов. Выбор цвета приводит к окраске в него формы (рис. 8.7).

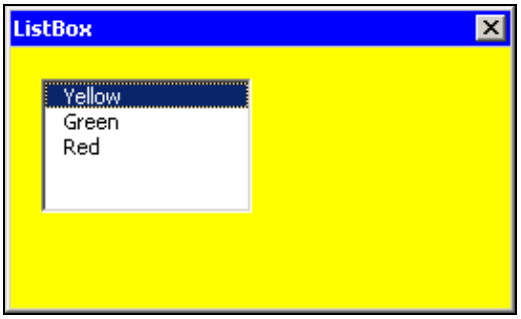


Рис. 8.7. Установка цвета формы

Листинг 8.10. Установка цвета формы

```
Private Sub UserForm_Initialize()
   ListBox1.AddItem "Yellow"
   ListBox1.AddItem "Green"
   ListBox1.AddItem "Red"
   ListBox1.ListIndex = 0
End Sub

Private Sub ListBox1_Change()
   Select Case ListBox1.ListIndex
      Case 0
         Me.BackColor = vbYellow
      Case 1
         Me.BackColor = vbGreen
      Case 2
         Me.BackColor = vbRed
   End Select
End Sub
```

Выбор нескольких элементов из списка

Свойство `MultiSelect` позволяет установить режим выбора нескольких элементов из списка. Допустимыми значениями этого свойства являются `fmMultiSelectSingle`, `fmMultiSelectMulti` и `fmMultiSelectExtended`.

В следующем примере демонстрируется его применение. На форме расположен список, поле и кнопка (рис. 8.8).

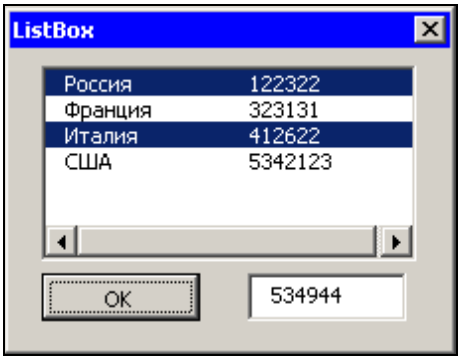


Рис. 8.8. Выбор нескольких элементов из списка

В список в виде двух столбцов выводятся данные о результатах продаж некоторой фирмы по странам, причем в первый столбец выводятся названия стран, а во второй — объемы продаж. При нажатии кнопки в поле выводится суммарный объем продаж по выбранным из списка странам. Используя окно **Properties**, установите им значения свойств, как показано в табл. 8.6, а в модуле формы наберите код из листинга 8.11.

Таблица 8.6. *Значения свойств элементов управления, установленные в окне* ***Properties***

Элемент управления	Свойство	Значение
Список	`Name`	`lstData`
Поле	`Name`	`txtRes`
Кнопка	`Name`	`cmdOK`
	`Caption`	`OK`

То, что список двухстолбцовый, устанавливается свойством `ColumnCount`. Ширина каждого из столбцов специфицируется свойством `ColumnWidths`. Элементы списка, как в совокупности, так и по отдельности устанавливаются и возвращаются свойством `List`. Определить, выбран ли специфицированный элемент списка, можно с помощью свойства `Selected`.

Листинг 8.11. Выбор нескольких элементов из списка

```
Private Sub UserForm_Initialize()
   txtRes.Locked = True
   lstData.ColumnCount = 2
```

```
    lstData.ColumnWidths = "70;70"
    lstData.MultiSelect = fmMultiSelectMulti
    Dim d(3, 1) As String
    d(0, 0) = "Россия": d(0, 1) = "122322"
    d(1, 0) = "Франция": d(1, 1) = "323131"
    d(2, 0) = "Италия": d(2, 1) = "412622"
    d(3, 0) = "США": d(3, 1) = "5342123"
    lstData.List() = d
End Sub

Private Sub cmdOK_Click()
    Dim i As Integer, s As Long
    s = 0
    For i = 0 To lstData.ListCount - 1
       If lstData.Selected(i) Then
          s = s + lstData.List(i, 1)
       End If
    Next
    If s > 0 Then txtRes.Text = CStr(s)
End Sub
```

Заполнение списка названиями месяцев

Список может быть заполнен на основе любого массива данных, включая названия месяцев. В последнем случае можно избежать рутинного поэлементного создания массива и воспользоваться оператором цикла. Дело

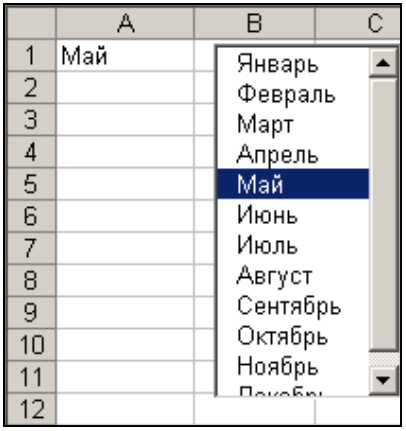


Рис. 8.9. Заполнение списка названиями месяцев

в том, что функция `DateSerial` возвращает строковое представление даты, а функция `Format` позволяет вывести дату по указанному формату, в частности, когда значение ее второго параметра, задающего формат, равно "mmmm", как раз и возвращается имя месяца. Для реализации данного проекта на рабочем листе расположите элемент управления **ListBox** (рис. 8.9), а в модуль **ЭтаКнига** и модуль рабочего листа введите код из листингов 8.12, *а*, *б*. Процедура, обрабатывающая событие `Open` объекта `Workbook`, реализует заполнение списка, а процедура, обрабатывающая событие `Click` объекта `ListBox1`, осуществляет вставку выбранного элемента из списка в ячейку **A1**.

Листинг 8.12, *а*. Заполнение списка названиями месяцев. Модуль ЭтаКнига

```
Private Sub Workbook_Open()
   Dim month(11) As String
   Worksheets(1).ListBox1.Clear
   For i = 1 To 12
      month(i - 1) = Format(DateSerial(2003, i, 1), "mmmm")
   Next
   Worksheets(1).ListBox1.List = month
End Sub
```

Листинг 8.12, *б*. Заполнение списка названиями месяцев. Модуль рабочего листа

```
Private Sub ListBox1_Click()
   Range("A1").Value = ListBox1.Value
End Sub
```

Поле со списком

Элемент управления **Поле со списком (ComboBox)** сочетает в себе черты элементов управления **ListBox** и **TextBox**, предназначен для отображения списка значений и позволяет, как вводить одно значение, так и выбирать одно значение из списка.

В следующем примере на форме расположены поле со списком и рисунок (рис. 8.10), а в модуле формы набран код из листинга 8.13. В список выводится два столбца: названия книг и имена файлов с изображениями их обложек. Так как информация о файлах является служебной, она не отображается в списке, для чего ширина второго столбца установлена равной 0. Выбор книги из списка приводит к отображению ее обложки на рисунке.

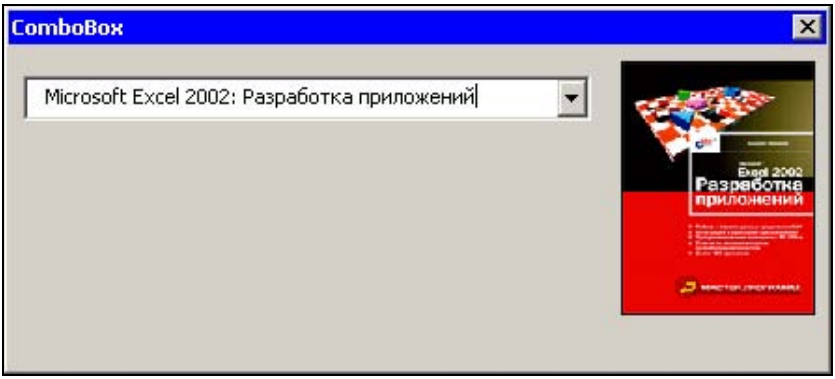


Рис. 8.10. Поле со списком, используемое для выбора одного значения

Листинг 8.13. Поле со списком, используемое для выбора одного значения

```
Dim bks(1, 1) As String
Dim path As String

Private Sub UserForm_Initialize()
   path = ThisWorkbook.path & "\"
   bks(0, 0) = "Microsoft Excel 2002: Разработка приложений"
   bks(0, 1) = "excel2002.jpg”
   bks(1, 0) = "Самоучитель Visual Studio .NET 2003"
   bks(1, 1) = "vsnet2003.jpg"

   ComboBox1.ColumnCount = 2
   ComboBox1.List = bks
   ComboBox1.ColumnWidths = "200;0"
   ComboBox1.ListIndex = 0
   Image1.Picture = _
      LoadPicture(path & ComboBox1.List(ComboBox1.ListIndex, 1))
   Image1.PictureSizeMode = fmPictureSizeModeStretch

End Sub

Private Sub ComboBox1_Change()
   Image1.Picture = _
      LoadPicture(path & ComboBox1.List(ComboBox1.ListIndex, 1))
End Sub
```

Рисунок

Элемент управления **Рисунок (Image)** предназначен для вывода растровых изображений, имеющих один из следующих форматов: BMP, CUR, GIF, ICO, JPG и WMF. Основными его свойствами являются свойство `Picture`, задающее ссылку на загружаемую картину, свойство `PictureSizeMode`, специфицирующее режим сжатия картинки и свойство `PictureAlignment`, определяющее ее местоположение. Следующий пример (листинг 8.14) демонстрирует применение этих свойств при загрузке изображения в рисунок (рис. 8.11).

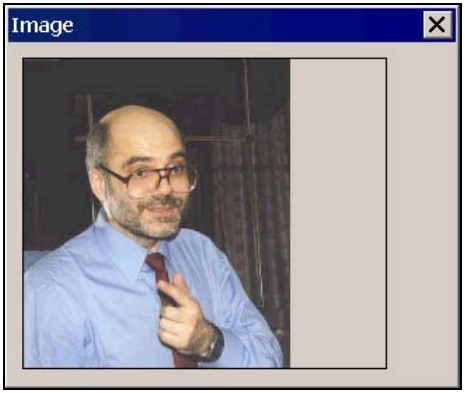


Рис. 8.11. Загрузка рисунка

Листинг 8.14. Загрузка рисунка

```
Private Sub UserForm_Initialize()
   Image1.Picture = LoadPicture("me.jpg")
   Image1.PictureSizeMode = fmPictureSizeModeZoom
   Image1.PictureAlignment = fmPictureAlignmentTopLeft
End Sub
```

Вставка в рисунок изображения выделенного диапазона

В качестве еще одного примера использования рисунка приведем приложение, в котором в рисунок вставляется изображение выбранного диапазона на рабочем листе. Создайте форму, на которой расположите кнопку и рисунок (рис. 8.12).

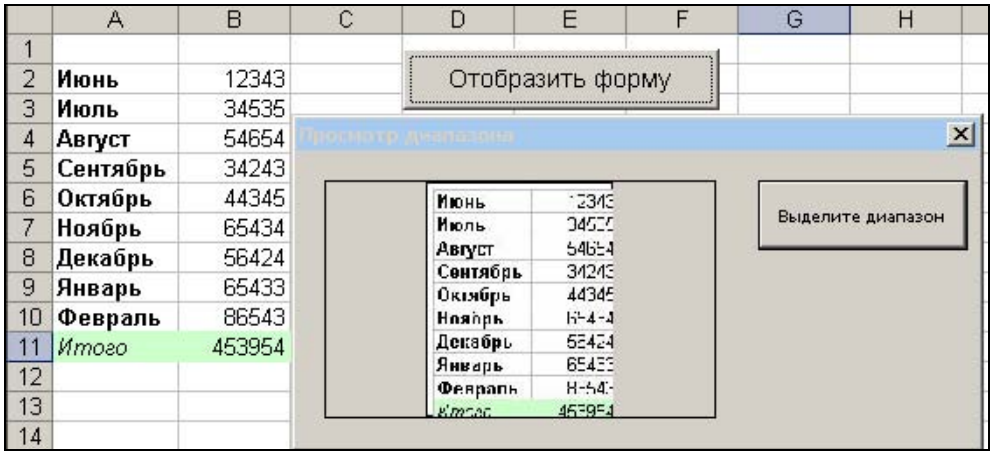


Рис. 8.12. Вставка в рисунок изображения выделенного диапазона

Используя окно **Properties**, установите у кнопки значения свойств `Name` и `Caption` равными `cmdSelect` и `Выделите диапазон`. В модуле формы наберите код из листинга 8.15, *а*. Кроме того, на рабочем листе расположите кнопку и, используя окно **Properties**, установите у этой кнопки значения свойств `Name` и `Caption` равными `cmdShowForm` и `Отобразить форму`. В модуле рабочего листа наберите код из листинга 8.15, *б*, отображающий форму в немодальном режиме, что позволяет пользователю выбрать диапазон без закрытия формы. Трюк, используемый в данном приложении, состоит в том, что на рабочем листе временно создается диаграмма, в которую копируется изображение диапазона. Затем диаграмма экспортируется в графический файл, содержание которого и загружается в рисунок.

Листинг 8.15, *а*. Вставка в рисунок изображения выделенного диапазона. Модуль формы

```
Private Sub cmdSelect_Click()
   Dim chrt As Chart
   Dim rgn As Range
   Set rgn = Application.InputBox("Выберете диапазон", Type:=8)
   rgn.CopyPicture xlScreen, xlBitmap
   Set chrt = ActiveSheet.ChartObjects.Add(0, 0, _
                        rgn.Width, rgn.Height).Chart
   With chrt
      .Paste
      .Export "Tmp.jpg"
```

```
      .Parent.Delete
   End With
   With Me.Controls("Image1")
      .Picture = LoadPicture("Tmp.jpg")
      .PictureSizeMode = fmPictureSizeModeZoom
   End With
   Kill "Tmp.jpg"
   Set rgn = Nothing
   Me.Repaint
End Sub
```

Листинг 8.15, *б*. Вставка в рисунок изображения выделенного диапазона. Модуль рабочего листа

```
Private Sub cmdShowForm_Click()
   UserForm1.Show False
End Sub
```

Флажок

Элемент управления **Флажок (CheckBox)** предлагает пользователю сделать выбор между двумя значениями такими, как Yes/No, True/False или On/Off. Текущее состояние флажка индицируется отображением специального знака наподобие галочки или отсутствием оного. Основными свойствами флажка являются свойство `Value`, идентифицирующее его состояние, и свойство `Caption`, задающее отображаемый при нем текст. В следующем примере на форме имеются два флажка и надпись. Флажки управляют стилем шрифта (курсив и полужирный) текста, отображаемого в надписи (рис. 8.13).

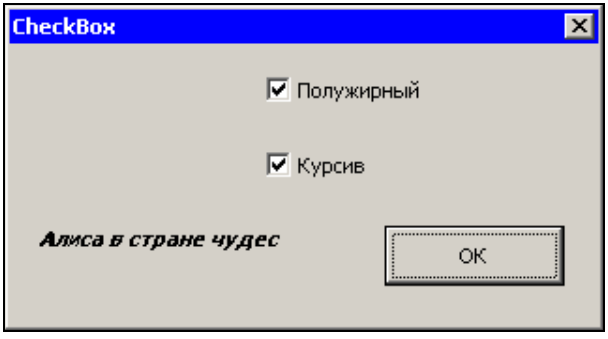


Рис. 8.13. Установка стиля шрифта

Для реализации этого проекта элементам управления, используя окно **Properties**, установите значения свойств, как показано в табл. 8.7, а в модуле формы наберите код из листинга 8.16.

Таблица 8.7. *Значения свойств элементов управления, установленные в окне* ***Properties***

Элемент управления	Свойство	Значение
Надпись	Name	lblTest
	Caption	Алиса в стране чудес
Флажок	Name	chkBold
	Caption	Полужирный
Флажок	Name	chkItalic
	Caption	Курсив
Кнопка	Name	cmdOK
	Caption	OK

Листинг 8.16. Установка стиля шрифта

```
Private Sub btnOK_Click()
   If chkBold.Value Then
      lblTest.Font.Bold = True
   Else
      lblTest.Font.Bold = False
   End If
   If chkItalic.Value Then
      lblTest.Font.Italic = True
   Else
      lblTest.Font.Italic = False
   End If
End Sub
```

Использование списка вместо группы флажков

Вместо группы флажков допустимо также использование списка **ListBox,** у которого значение свойства `ListStyle` установлено равным `fmListStyleOption`, а значение свойства `MultiSelect` равным `fmMultiSelectMulti` или `fmMultiSelectExtended`. Код из листинга 8.17 демонстрирует подобный подход. В нем вместо двух флажков из предыдущего примера (рис. 8.14)

располагается один список, у которого значение свойства `Name` установлено равным `lstFont`.

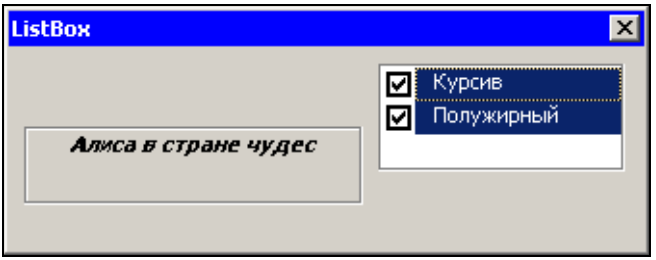


Рис. 8.14. Использование **ListBox** в качестве группы флажков

Листинг 8.17. Использование ListBox в качестве группы флажков

```
Private Sub UserForm_Initialize()
   lstFont.MultiSelect = fmMultiSelectMulti
   lstFont.ListStyle = fmListStyleOption
   lstFont.AddItem "Курсив"
   lstFont.AddItem "Полужирный"
   lblTest.SpecialEffect = fmSpecialEffectEtched
   lblTest.TextAlign = fmTextAlignCenter
End Sub

Private Sub lstFont_Change()
   If lstFont.Selected(0) Then
      lblTest.Font.Italic = True
   Else
      lblTest.Font.Italic = False
   End If
   If lstFont.Selected(1) Then
      lblTest.Font.Bold = True
   Else
      lblTest.Font.Bold = False
   End If
End Sub
```

Изменение цвета выделенного диапазона

Приведем еще один пример использования флажка, но в этот раз для изменения цвета выделенного диапазона, а именно выделенный диапазон теперь будет окрашиваться в красный цвет. Для этого расположите на рабочем листе флажок. Установите значение его свойства `Caption` равным `Окрасить`, а в модуле **ЭтаКнига** наберите код из листинга 8.18. Вот и все.

Листинг 8.18. Изменение цвета выделенного диапазона

```
Private Sub Workbook_SheetSelectionChange(ByVal Sh As Object, _
        ByVal Target As Range)
   Static OldRange As Range
   If Not OldRange Is Nothing Then
      OldRange.Interior.ColorIndex = xlColorIndexNone
   End If
   If Worksheets(1).CheckBox1.Value Then
      Target.Interior.Color = vbRed
   End If
   Set OldRange = Target
End Sub
```

Рамка

Элемент управления **Рамка (Frame)** используется для визуальной группировки других элементов управления. Основным его свойством является свойство `Caption`, задающее текст пояснительной надписи.

Переключатель

Элемент управления **Переключатель (OptionButton)** предлагает пользователю сделать выбор одного элемента из группы альтернативных возможностей. Переключатели, размещенные в форме или в элементе управления **Frame** как в контейнере, являются взаимно исключающими. Основными свойствами переключателя являются свойство `Value`, идентифицирующее его состояние, и свойство `Caption`, задающее отображаемый при нем текст. В следующем примере три переключателя расположены в рамке. Выбор переключателя вызывает смену курсора. У переключателей отображаются названия курсоров и их изображения. Для реализации данного проекта потребуются три файла курсоров hnodrop.cur, hnwse.cur и larrow.cur, которые можно найти в каталоге C:\Windows\Cursors, а также эти же файлы, но в формате GIF.

Итак, сконструируйте форму, на которой расположите рамку и три переключателя (рис. 8.15).

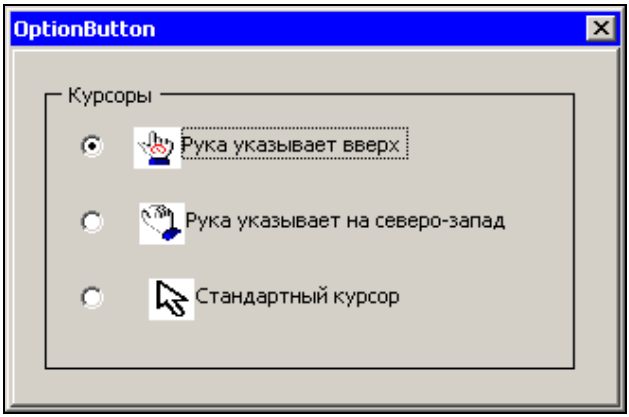


Рис. 8.15. Установка курсора при помощи переключателей

Используя окно **Properties**, установите значения свойств элементов управления, как показано в табл. 8.8, а в модуле формы наберите код из листинга 8.19. Свойство `Picture` позволит дополнительно к тексту вывести в переключатель картинку. Свойство `PicturePosition` устанавливает местоположение картинки. Свойство `MousePointer` задает тип курсора. Если его значение равно `fmMousePointerCustom`, то возможно создать пользовательский курсор на основе CUR файла, ссылка на который задается свойством `MouseIcon`.

Таблица 8.8. *Значения свойств элементов управления, установленные в окне* ***Properties***

Элемент управления	Свойство	Значение
Рамка	`Caption`	`Курсоры`
Переключатель	`Caption`	`Рука указывает вверх`
	`Picture`	`Ссылка на файл hnodrop.gif`
	`PicturePosition`	`fmPicturePositionLeftCenter`
Переключатель	`Caption`	`Рука указывает на северо-запад`
	`Picture`	`Ссылка на файл hnwse.gif`
	`PicturePosition`	`fmPicturePositionLeftCenter`
Переключатель	`Caption`	`Стандартный курсор`
	`Picture`	`Ссылка на файл larrow.cur`
	`PicturePosition`	`fmPicturePositionLeftCenter`

Листинг 8.19. Установка курсора при помощи переключателей

```
Private Path As String

Private Sub UserForm_Initialize()
   Path = ThisWorkbook.Path & "\"
   Me.MousePointer = fmMousePointerCustom
   OptionButton3.Value = True
End Sub

Private Sub OptionButton1_Click()
   Me.MouseIcon = LoadPicture(Path & "hnodrop.cur")
End Sub

Private Sub OptionButton2_Click()
   Me.MouseIcon = LoadPicture(Path & "hnwse.cur")
End Sub

Private Sub OptionButton3_Click()
   Me.MouseIcon = LoadPicture(Path & "larrow.cur")
End Sub
```

Вместо группы переключателей допустимо также использование списка **ListBox,** у которого значение свойства `ListStyle` установлено равным `fmListStyleOption`, а значение свойства `MultiSelect` — равным `fmMultiSelectSingle`.

Запуск и остановка часов при помощи переключателя

В качестве еще одной иллюстрации приведем пример, в котором переключатели расположены на рабочем листе и управляют работой часов. Итак, на рабочем листе расположите два переключателя. Используя окно **Properties**, установите им значения свойств `Name` и `Caption` равными `optStart` и `Start`, и `optStop` и `Stop`. В модуле рабочего листа и стандартном модуле наберите код из листингов 8.20, *a, b*. При выборе переключателя **Start** в ячейке **A1** отображается текущее время, а при выборе переключателя **Stop** время останавливается. Работа часов реализуется с помощью метода `OnTime` объекта `Application`, который устанавливает выполнение специфицированного макроса на указанное время.

Листинг 8.20, *а*. Запуск и остановка часов. Модуль рабочего листа

```
Private Sub optStart_Click()
   StartTime
End Sub

Private Sub optStop_Click()
   StopTime
End Sub
```

Листинг 8.20, *б*. Запуск и остановка часов. Стандартный модуль

```
Private NewTime As Date

Sub StartTime()
    NewTime = Time + TimeValue("00:00:01")
    Application.OnTime NewTime, "RefreshTime"
End Sub

Sub RefreshTime()
   Range("A1").Value = Time
   StartTime
End Sub

Sub StopTime()
   On Error Resume Next
   Application.OnTime EarliestTime:=NewTime, _
                      Procedure:="RefreshTime", _
                      Schedule:=False
End Sub
```

Выключатель

Элемент управления **Выключатель** (**ToggleButton**) выполняет те же функции, что и флажок, только визуально он выглядит как кнопка, которая может находиться в двух состояниях: выбранном и невыбранном. В следующем примере демонстрируется применение выключателя при контроле над возможностью ввода данных в поле ввода. При выбранном выключателе ввод блокируется и на нем отображается надпись `Запрет ввода`. При невыбранном

выключателе ввод разрешен и в нем отображается надпись Ввод. Итак, создайте форму, на которой расположите надпись, поле и выключатель (рис. 8.16).

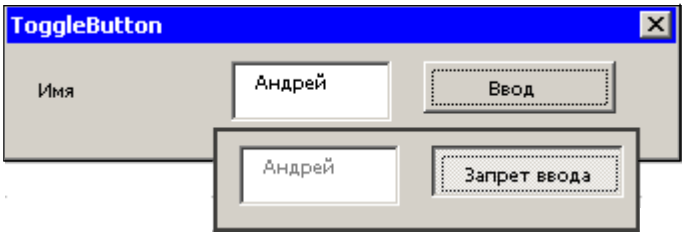


Рис. 8.16. Управление вводом данных

Используя окно **Properties**, установите значения свойств элементов управления, как показано в табл. 8.9, а в модуле формы наберите код из листинга 8.21.

Таблица 8.9. *Значения свойств элементов управления, установленные в окне* ***Properties***

Элемент управления	Свойство	Значение
Надпись	Caption	Имя
Выключатель	Name	tglInput
Поле	Name	txtName

Листинг 8.21. Управление вводом данных

```
Private Sub UserForm_initialize()
   tglInput.Value = False
End Sub

Private Sub tglInput_Change()
   txtName.Enabled = Not tglInput.Value
   If tglInput.Value Then
      tglInput.Caption = "Запрет ввода"
   Else
      tglInput.Caption = "Ввода"
   End If
End Sub
```

Полоса прокрутки

Элемент управления **Полоса прокрутки** (**ScrollBar**) предлагает установить числовое значение, основываясь на положении ползунка. Основными свойствами полосы прокрутки являются свойства `Min`, `Max` и `Value`, специфицирующие минимальное, максимальное и текущее ее значения. Основным событием — событие `Change`, генерируемое при изменении текущего значения полосы.

В следующем примере демонстрируется применение полос прокрутки для установки размеров рисунка. На форме имеется рисунок и две полосы прокрутки (рис. 8.17). Используя окно **Properties**, установите значения свойств элементов управления, как показано в табл. 8.10, а в модуле формы наберите код из листинга 8.22. Свойства `InsideWidth` и `InsideHeight` формы возвращают ширину и высоту пользовательской части. Свойство `BackStyle` рисунка может принимать два значения: `fmBackStyleTransparent` и `fmBackStyleOpaque`, что устанавливает, является ли его фон прозрачным.

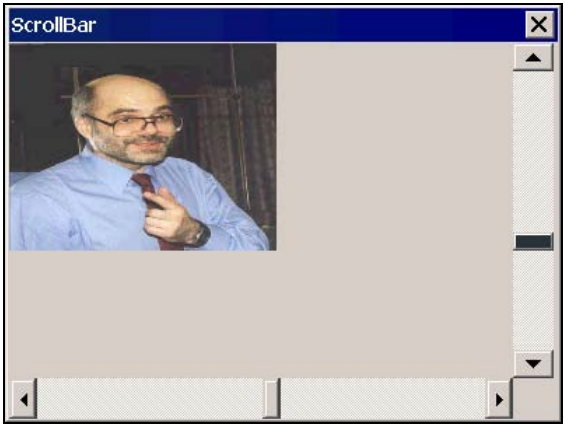


Рис. 8.17. Изменение размеров рисунка с помощью полосы прокрутки

Таблица 8.10. *Значения свойств элементов управления, установленные в окне* ***Properties***

Элемент управления	Свойство	Значение
Рисунок	`Name`	`imgPic`
Полоса прокрутки	`Name`	`scrHBar`
Полоса прокрутки	`Name`	`scrWBar`

Листинг 8.22. Изменение размеров рисунка с помощью полосы прокрутки

```
Private Sub UserForm_Initialize()
   imgPic.Top = 0
   imgPic.Left = 0
   imgPic.BorderStyle = fmBorderStyleNone
   imgPic.BackStyle = fmBackStyleTransparent
   imgPic.PictureSizeMode = fmPictureSizeModeStretch
   imgPic.Picture = LoadPicture(ThisWorkbook.Path & "\me.gif")
   scrHBar.Min = 0
   scrHBar.Max = Me.InsideHeight - scrWBar.Height
   scrHBar.Value = imgPic.Height
   scrWBar.Min = 0
   scrWBar.Max = Me.InsideWidth - scrHBar.Width
   scrWBar.Value = imgPic.Width
End Sub

Private Sub scrHBar_Change()
   imgPic.Height = scrHBar.Value
End Sub

Private Sub scrWBar_Change()
   imgPic.Width = scrWBar.Value
End Sub
```

Счетчик

Элемент управления **Счетчик** (**SpinButton**) можно рассматривать, как полосу прокрутки без ползунка, и он выполняет те же функции, что и полоса прокрутки.

В следующем примере создается простейшее приложение по вводу данных (имя, возраст и пол) о клиенте в таблицу рабочего листа (рис. 8.18).

Создайте форму, на которой расположите три надписи, три поля, два счетчика и кнопку. Используя окно **Properties**, установите им значения свойств, как показано в табл. 8.11. В модуль формы введите код из листинга 8.23.

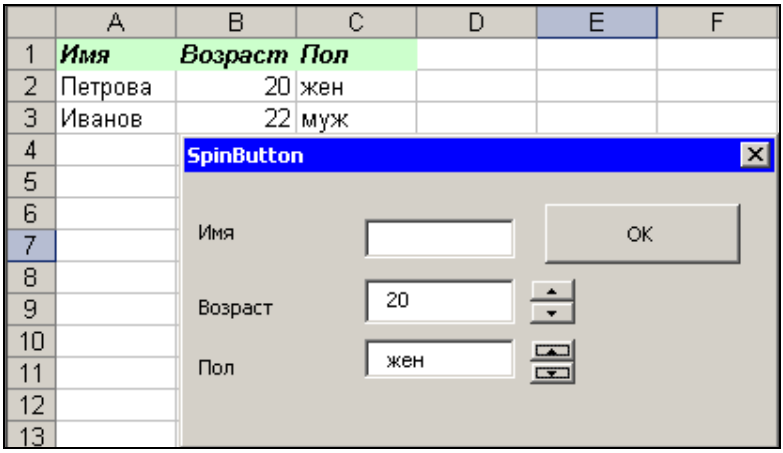


Рис. 8.18. Ввод значений из формы в рабочий лист

Таблица 8.11. *Значения свойств элементов управления, установленные в окне* ***Properties***

Элемент управления	Свойство	Значение
Надпись	`Caption`	`Имя`
Надпись	`Caption`	`Возраст`
Надпись	`Caption`	`Пол`
Поле	`Name`	`txtName`
Поле	`Name`	`txtAge`
Поле	`Name`	`txtSex`
Счетчик	`Name`	`spnAge`
Счетчик	`Name`	`spnSex`
Кнопка	`Name`	`cmdOK`
	`Caption`	`OK`

В ячейки **A1**, **B1** и **C1** рабочего листа введите `Имя`, `Возраст` и `Пол`. Первый счетчик вводит в поле **Возраст** целые значения от 0 до 100, а второй счетчик — в поле **Пол** два значения: `Муж` и `Жен`. Поля **Возраст** и **Пол** заблокированы для прямого ввода в них значений пользователем. Номер очередной пустой строки в таблице определяется при помощи функции рабочего листа `CountA`, возвращающей число непустых ячеек в указанном диапазоне.

Листинг 8.23. Заполнение табличной базы данных

```
Enum sex
   male = 0
   female = 1
End Enum

Private Sub UserForm_Initialize()
   spnAge.Min = 0
   spnAge.Max = 100
   spnAge.Value = 20
   txtAge.Locked = True
   spnSex.Min = 0
   spnSex.Max = 1
   spnSex.Value = 1
   txtSex.Locked = True
End Sub

Private Sub spnAge_Change()
  txtAge.Text = CStr(spnAge.Value)
End Sub

Private Sub spnSex_Change()
   txtSex.Text = Switch(spnSex.Value = sex.male, "Муж", _
                        spnSex.Value = sex.female, "Жен")
End Sub

Private Sub cmdOK_Click()
   Dim n As Integer
   n = Application.WorksheetFunction.CountA(Range("A:A")) + 1
   Cells(n, 1).Value = txtName.Text
   Cells(n, 2).Value = txtAge.Text
   Cells(n, 3).Value = txtSex.Text
End Sub
```

Как можно перебирать элементы управления

Элементы управления образуют семейство `Controls` и поэтому их можно перебирать последовательно в цикле. Идентификация элемента управления может производиться по типу с помощью функции `TypeName` и по значению либо свойства `Tag`, либо `Name`. Например, в следующем проекте в форме имеются четыре поля `TextBox1`, , `TextBox4` и кнопка `CommandButton1`. В поля `TextBox1`, , `TextBox3` вводятся какие-либо числа, а в поле `TextBox4` по нажатию кнопки отображается их сумма. Вместо того чтобы использовать код из листинга 8.24, *а,* можно применить подход из листинга 8.24, *б*, выигрышность которого становится тем очевиднее, чем больше имеется элементов управления.

Листинг 8.24, *а*. Явная ссылка на имена элементов управления

```
Private Sub CommandButton1_Click()
   Dim a1 As Double, a2 As Double, a3 As Double, a4 As Double
   Dim a As Double
   a1 = CDbl(TextBox1.Text)
   a2 = CDbl(TextBox2.Text)
   a3 = CDbl(TextBox3.Text)
   a4 = CDbl(TextBox4.Text)
   a = a1 + a2 + a3 + a4
   TextBox4.Text = CStr(a)
End Sub
```

Листинг 8.24, *б*. Перебор элементов управления

```
Private Sub CommandButton1_Click()
   Dim c As Control
   Dim s As Double
   s = 0
   For Each c In Me.Controls
     If TypeName(c) = "TextBox" And c.Name <> "TextBox4" Then
        s = s + c.Text
     End If
   Next
   TextBox4.Text = CStr(s)
End Sub
```

Обеспечение функционирования кнопки только в случае заполненных полей

При проектировании приложений для большего контроля над действиями пользователя часто возникает необходимость блокировать кнопки (или какие-то другие элементы управления) до заполнения пользователем формы, и только по ее заполнению кнопки разблокировать. Продемонстрируем на примере, как это можно сделать. На форме имеются три поля и кнопка. В первые два поля вводятся слагаемые, а в третье, по нажатию кнопки, их сумма. Третье поле может получать фокус, но оно блокировано для пользователя, т. е. он не может вводить в него значения. Это обеспечивается установкой значения свойства `Locked` равным `True`. Кнопка находится в незаблокированном состоянии в случае, если первое и второе поля заполнены. Блокировкой элемента управления с тем, чтобы он даже не мог получать фокус, управляет свойство `Enabled`. В приложении нам надо отслеживать ввод данных пользователем в поля. Здесь на помощь приходит процедура обработки события `KeyUp` поля, которое генерируется в случае, если поле имеет фокус и пользователь отпускает клавишу клавиатуры (листинг 8.25). В этой процедуре надо отследить, не пусты ли оба поля и если они не пусты, разблокировать кнопку. В противном случае, кнопка должна быть заблокирована.

Листинг 8.25. Обеспечение функционирования кнопки только в случае заполненных полей

```
Private Sub UserForm_Initialize()
   CommandButton1.Enabled = False
   TextBox3.Locked = True
End Sub

Private Sub ButtonState()
   If (Trim(TextBox1.Text) = "") Or (Trim(TextBox2.Text) = "") Then
      CommandButton1.Enabled = False
   Else
      CommandButton1.Enabled = True
   End If
End Sub

Private Sub TextBox1_KeyUp(ByVal KeyCode As MSForms.ReturnInteger, _
            ByVal Shift As Integer)
   ButtonState
```

```
End Sub

Private Sub TextBox2_KeyUp(ByVal KeyCode As MSForms.ReturnInteger, _
            ByVal Shift As Integer)
   ButtonState
End Sub

Private Sub CommandButton1_Click()
   TextBox3.Text = CStr(CDbl(TextBox1.Text) + CDbl(TextBox2.Text))
End Sub
```

Обмен сообщениями между формами

При обмене сообщениями между формами можно определить открытые переменные, которые и будут передавать искомые значения. Но можно пойти другим способом, воспользовавшись семейством `Controls`, которое содержит в себе все элементы управления данной формы, и доступ к которым возможен либо по индексу, либо по имени. Продемонстрируем данный подход на примере (листинги 8.26, *а*, *б*, *в*). В проекте имеются две формы: **UserForm1** и **UserForm2** (рис. 8.19), причем:

- ❒ отображение первой формы производится в полимодальном режиме при нажатии кнопки, размещенной на рабочем листе;

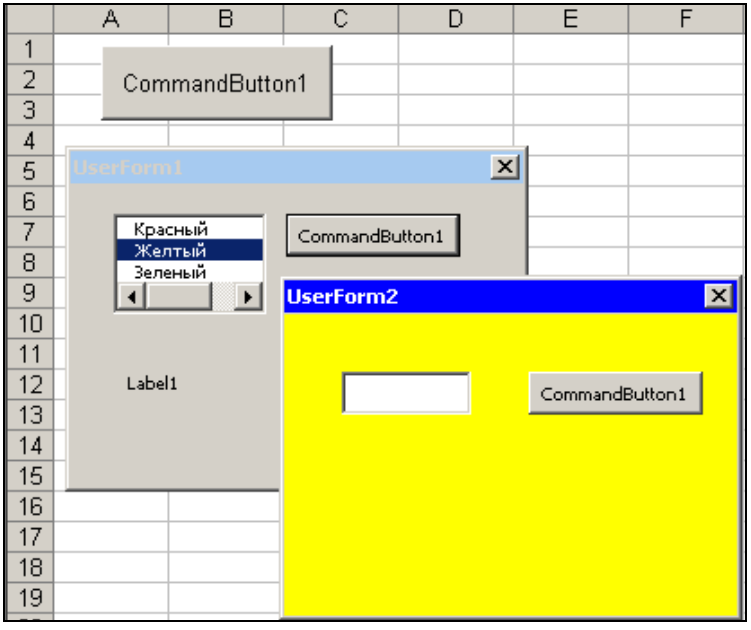


Рис. 8.19. Обмен сообщениями между формами

- в форме **UserForm1** имеются список, надпись и кнопка. В списке выводится перечень цветов. Выбор цвета и нажатие кнопки приводит к отображению второго окна, у которого цвет фона формы будет установлен в соответствии с выбранным цветом;
- в форме **UserForm2** имеются поле и кнопка. Нажатие кнопки вызывает отображение первой формы, у которой в надпись будет выведена строка, являющаяся конкатенацией строк `Hello`, и той, которая введена пользователем в поле.

Листинг 8.26, *а*. Обмен сообщениями между формами. Модуль формы `UserForm1`

```
Private Sub UserForm_Initialize()
   Dim vls(2, 1) As Variant
   vls(0, 0) = "Красный": vls(0, 1) = vbRed
   vls(1, 0) = "Желтый": vls(1, 1) = vbYellow
   vls(2, 0) = "Зеленый": vls(2, 1) = vbGreen
   ListBox1.ColumnCount = 2
   ListBox1.TextColumn = 2
   ListBox1.ColumnWidths = "80;0"
   ListBox1.BoundColumn = 2
   ListBox1.List = vls
End Sub

Private Sub CommandButton1_Click()
   UserForm2.Show vbModeless
   UserForm2.BackColor = ListBox1.Value
   UserForm2.Top = UserForm1.Top + 100
   UserForm2.Left = UserForm1.Left + 100
End Sub
```

Листинг 8.26, *б*. Обмен сообщениями между формами. Модуль формы `UserForm2`

```
Private Sub CommandButton1_Click()
   UserForm1.Controls("Label1").Caption = "Hello, " & TextBox1.Text
   UserForm1.Show vbModeless
   UserForm1.Top = UserForm2.Top + 100
   UserForm1.Left = UserForm2.Left + 100
End Sub
```

Листинг 8.26, *в*. Обмен сообщениями между формами. Модуль рабочего листа

```
Private Sub CommandButton1_Click()
   UserForm1.Show vbModeless
End Sub
```

Объект *DataObject*

Объект `DataObject` используется для инкапсуляции текстовых данных при их перемещении. Продемонстрируем работу с этим объектом на простом примере. На форме имеются два поля и кнопка (рис. 8.20). В поля разрешен многострочный ввод данных. При нажатии кнопки выделенные данные из первого поля копируются во второе.

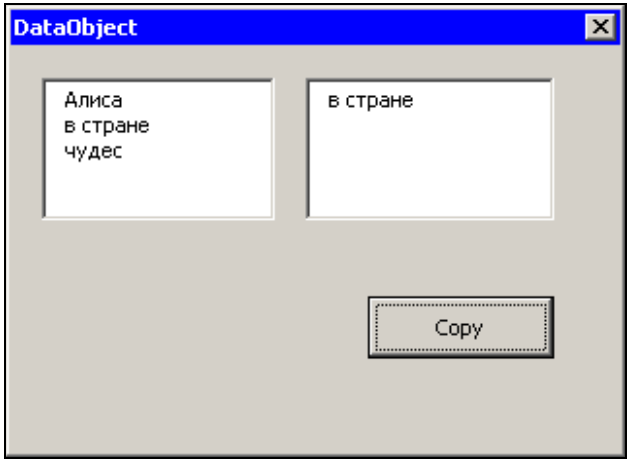


Рис. 8.20. Перемещение текстовых данных между полями

Итак, создайте форму, на которой расположите указанные элементы управления. Используя окно **Properties**, установите им значения свойств, как показано в табл. 8.12. В модуле формы наберите код из листинга 8.27.

Таблица 8.12. *Значения свойств элементов управления, установленные в окне* ***Properties***

Элемент управления	Свойство	Значение
Поле	`Name`	`txtSource`
	`MultiLine`	`True`

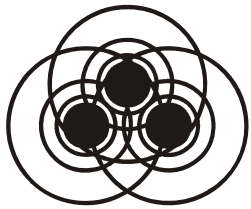
Таблица 8.12 *(окончание)*

Элемент управления	Свойство	Значение
Поле	Name	TxtTarget
	MultiLine	True
Кнопка	Name	cmdCopy
	Caption	Copy

Свойство `MultiLine` поля устанавливает многострочный режим вывода текста. В следующем коде метод `Copy` поля копирует в буфер обмена выделенный фрагмент текста, а свойство `SelText` его возвращает. Метод `GetFromClipboard` объекта `DataObject` забирает данные из буфера обмена, а метод `GetText`, при значении его параметра равным 1, возвращает данные из объекта `DataObject` в текстовом формате.

Листинг 8.27. Перемещение текстовых данных между полями

```
Dim d As DataObject

Private Sub UserForm_Initialize()
   Set d = New DataObject
   txtSource.Text = "Алиса" _
        & vbCr & "в стране" _
        & vbCr & "чудес"
End Sub

Private Sub cmdCopy_Click()
   If Len(txtSource.SelText) > 0 Then
      txtSource.Copy
      d.GetFromClipboard
      txtTarget.Text = d.GetText(1)
   Else
      MsgBox "Выделите что-нибудь в первом поле!"
   End If
End Sub
```



Глава 9

Создание меню, контекстного меню и панели инструментов

В предыдущих главах было приведено огромное количество примеров создания приложений, позволяющих автоматизировать и существенно упростить работу в MS Office. Одной из наиболее важных задач при разработке приложения является создание собственного интерфейса, при этом необходимо исключить ненужные панели инструментов, создать пользовательские панели инструментов и преобразовать строку меню под нужды приложения. Как раз одним из достоинств MS Office является то, что его интерфейс легко настроить под конкретное приложение, создавая и настраивая панели инструментов и строки меню. Панели инструментов, строки меню и контекстные меню (вызываемые щелчком правой кнопки мыши) фактически являются одинаковыми объектами, для которых в интерфейсе пользователя используется термин *панели инструментов*, а в VBA — термин *панели команд* (*command bar*). Данные о них инкапсулируются в объектах `CommandBar`, которые в своей совокупности образуют семейство `CommandBars`.

Создание строки меню

Данные о меню, контекстном меню или панели инструментов инкапсулируются в объектах `CommandBar`, которые в совокупности образуют семейство `CommandBars`. Продемонстрируем технику построения меню на простом примере. Меню будет иметь следующую структуру. Выпадающее меню **Demo menu**, в котором имеются четыре пункта (**Menu Item1**, **Menu Item2**, **Submenu1** и **Submenu Item2 On/Off**) и два разделителя. У пункта меню **Submenu1** имеется выпадающее подменю, состоящее из трех пунктов (**Submenu Item1**, **Submenu Item2** и **SubSubMenu1**) и разделитель. У пункта меню **SubSubMenu1** имеется выпадающее подменю, состоящее из двух пунктов (**SubSubMenu Item1** и **SubSubMenu Item2**).

Пункты **Submenu Item2 On/Off** и **Submenu Item1** имеют не только текст, но и значки  и  . Пункты **SubSubMenu Item1** и **SubSubMenu Item2** кроме текста

имеют значки с номерами этих пунктов в выпадающем меню. При выборе каждой из команд меню, за исключением **Submenu Item2 On/Off**, для подтверждения действий пользователя, выполняются *процедуры-заглушки*, которые лишь обозначают присутствие кода. Выбор же команды **Submenu Item2 On/Off** действительно производит включение/отключение пункта **Submenu Item2** (рис. 9.1).

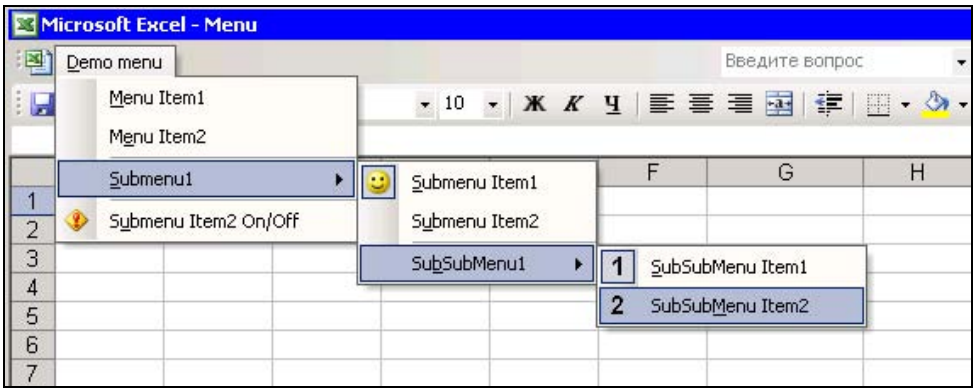


Рис. 9.1. Строка меню

В данном проекте код набирается в двух модулях: **ЭтаКнига** и стандартном.

- В модуле **ЭтаКнига** (листинг 9.1, *а*) имеются две процедуры. Первая из них обрабатывает событие `Open` объекта `Workbook`. Эта процедура инициализирует открытую переменную `CommandBarName`, в которой задано имя строки меню, и вызывает процедуру `MenuBuilder`, которая создает пользовательское меню. Строка меню удалялась при закрытии книги. Для этого в коде обрабатывается событие `BeforeClose` объекта `Workbook`. В процедуре обработки этого события вызывается процедура `MenuKiller`, которая как раз и удаляет пользовательскую строку меню.
- В модуле рабочего листа (листинг 9.1, *б*) объявлена открытая переменная `CommandBarName` и определены процедуры `MenuBuilder` и `MenuKiller`. Кроме того, в нем определены процедуры `MacroForMenuItem1`, `MacroForMenuItem2`, `MacroForSubMenuItem1`, `MacroForSubMenuItem2`, `MacroForSubSubMenuItem1`, `MacroForSubSubMenuItem2`, `MenuItemSwitcher`, которые реализуют действия, связанные с выбором соответствующих пунктов меню.

Строка меню является объектом `CommandBar`. Конструируется она методом `Add` семейства `CommandBars`, при этом значение параметра `MenuBar` этого метода должно быть установлено равным `True`. Параметр `Temporary` определяет, является ли строка меню временной и удаляется при закрытии всего приложения, но не рабочей книги. Параметр `Position` задает местоположение

строки меню. Если его значение равно `msoBarTop`, то она выводится вверху окна. Другими допустимыми значениям этого параметра являются `msoBarLeft`, `msoBarRight`, `msoBarBottom`. Параметр `Name` определяет имя строки меню.

```
Dim cb As CommandBar
Set cb = Application.CommandBars.Add(Name:=CommandBarName, _
                                     Position:=msoBarTop, _
                                     MenuBar:=True, _
                                     Temporary:=True)
```

Выпадающее меню является объектом `CommandBarControl`. Конструируется оно методом `Add` семейства `Controls`. При этом значение параметра `Type` этого метода должно быть установлено равным `msoControlPopup`, потому что семейство `Controls` содержит не только выпадающие меню, но и пункты меню. Параметр `Temporary` определяет, является ли выпадающее меню или пункт временным. Свойство `Caption` задает заголовок выпадающего меню.

```
Dim cbMenu As CommandBarControl
Set cbMenu = cb.Controls.Add(Type:=msoControlPopup, _
                             Temporary:=True)
```

Пункт меню создается методом `Add`, у которого значение параметра `Type` установлено равным `msoControlButton`. Свойства `Caption` и `OnAction`, задают его заголовок и процедуру, которая выполняется при его выборе.

```
With cbMenu.Controls.Add(Type:=msoControlButton, _
                         Temporary:=True)
   .Caption = "&Menu Item1"
   .OnAction = "MacroForMenuItem1"
End With
```

Если в пункте меню надо кроме заголовка вывести и значок, то у него устанавливается значение свойства `Style` равным `msoButtonIconAndCaption`, а в качестве значения свойства `FaceId` — идентификатор стандартного значка, используемого в Microsoft Office. Для конструирования разделительной линии, у пункта, перед которым она создается, надо установить значение свойства `BeginGroup` равным `True`.

Листинг 9.1, *а*. Строка меню. Модуль ЭтаКнига

```
Option Explicit

Private Sub Workbook_Open()
   CommandBarName = "MyCommandBarName"
   MenuBuilder
```

```
End Sub

Private Sub Workbook_BeforeClose(Cancel As Boolean)
   MenuKiller
End Sub
```

Листинг 9.1, *б*. Строка меню. Стандартный модуль

```
Option Explicit

Public CommandBarName As String ' Имя строки меню
Private cbcItem As CommandBarControl

Sub MenuBuilder()
   Dim cb As CommandBar
   Dim cbMenu As CommandBarControl, cbSubMenu As CommandBarControl
   ' Удаление пользовательского меню, если оно уже существует
   MenuKiller
   ' Создание строки меню, замещающей встроенное меню
   Set cb = Application.CommandBars.Add( _
                          Name:=CommandBarName, _
                          Position:=msoBarTop, _
                          MenuBar:=True, _
                          Temporary:=True)
   ' Создание выпадающего меню
   Set cbMenu = cb.Controls.Add( _
                           Type:=msoControlPopup, _
                           Temporary:=True)
   With cbMenu
      .Caption = "&Demo menu"
      .BeginGroup = False
   End With

   ' Если меню отсутствует, то выход из программы
   If cbMenu Is Nothing Then Exit Sub

   ' Добавление пункта меню
   With cbMenu.Controls.Add( _
                       Type:=msoControlButton, _
```

```
                    Temporary:=True)
   .Caption = "&Menu Item1"
   .OnAction = "MacroForMenuItem1"
End With

' Добавление пункта меню
With cbMenu.Controls.Add( _
                    Type:=msoControlButton, _
                    Temporary:=True)
   .Caption = "M&enu Item2"
   .OnAction = "MacroForMenuItem2"
End With

' Добавление выпадающего подменю первого уровня
Set cbSubMenu = cbMenu.Controls.Add( _
                                Type:=msoControlPopup, _
                                Temporary:=True)
With cbSubMenu
   .Caption = "&Submenu1"
   .BeginGroup = True
End With

' Добавление пункта меню
With cbMenu.Controls.Add( _
                    Type:=msoControlButton, _
                    Temporary:=True)
   .Caption = "S&ubmenu Item2 On/Off"
   .OnAction = "MenuItemSwitcher"
   .Style = msoButtonIconAndCaption
   .FaceId = 463
   .BeginGroup = True
End With

' Добавление пункта меню в выпадающее подменю первого уровня
With cbSubMenu.Controls.Add( _
                        Type:=msoControlButton, _
                        Temporary:=True)
   .Caption = "&Submenu Item1"
   .OnAction = "MacroForSubMenuItem1"
```

```
    .Style = msoButtonIconAndCaption
    .FaceId = 2950
    .State = msoButtonDown
End With

' Добавление пункта меню в выпадающее подменю первого уровня
Set cbcItem = cbSubMenu.Controls.Add( _
                                Type:=msoControlButton, _
                                Temporary:=True)
With cbcItem
    .Caption = "S&ubmenu Item2"
    .OnAction = "MacroForSubMenuItem2"
    .Enabled = False
End With

' Добавление выпадающего подменю второго уровня
Set cbSubMenu = cbSubMenu.Controls.Add( _
                                  Type:=msoControlPopup, _
                                  Temporary:=True)
With cbSubMenu
    .Caption = "Su&bSubMenu1"
    .BeginGroup = True
End With

' Добавление пункта меню в выпадающее подменю второго уровня
With cbSubMenu.Controls.Add( _
                       Type:=msoControlButton, _
                       Temporary:=True)
    .Caption = "&SubSubMenu Item1"
    .OnAction = "MacroForSubSubMenuItem1"
    .Style = msoButtonIconAndCaption
    .FaceId = 71
    .State = msoButtonDown
End With

' Добавление пункта меню в выпадающее подменю второго уровня
With cbSubMenu.Controls.Add( _
                      Type:=msoControlButton, _
                      Temporary:=True)
```

```
        .Caption = "SubSub&Menu Item2"
        .OnAction = "MacroForSubSubMenuItem2"
        .Style = msoButtonIconAndCaption
        .FaceId = 72
        .Enabled = True
    End With

    ' Отобразить меню
    cb.Visible = True
    Set cbSubMenu = Nothing
    Set cbMenu = Nothing
    Set cb = Nothing
End Sub

Sub MenuKiller()
    ' Удаление строки меню
    On Error Resume Next
    Application.CommandBars(CommandBarName).Delete
    On Error GoTo 0
End Sub

Sub MacroForMenuItem1()
    MsgBox "This is Menu Item 1!", vbInformation, ThisWorkbook.Name
End Sub

Sub MacroForMenuItem2()
    MsgBox "This is Menu Item 2!", vbInformation, ThisWorkbook.Name
End Sub

Sub MacroForSubMenuItem1()
    MsgBox "This is Sub Menu Item 1!", _
           vbInformation, ThisWorkbook.Name
End Sub

Sub MacroForSubMenuItem2()
    MsgBox "This is Sub Menu Item 2!", _
           vbInformation, ThisWorkbook.Name
End Sub
```

```
Sub MacroForSubSubMenuItem1()
   MsgBox "This is Sub Sub Menu Item 1!", _
          vbInformation, ThisWorkbook.Name
End Sub

Sub MacroForSubSubMenuItem2()
   MsgBox "This is Sub Sub Menu Item 2!", _
          vbInformation, ThisWorkbook.Name
End Sub

Sub MenuItemSwitcher()
   cbcItem.Enabled = Not cbcItem.Enabled
End Sub
```

Типичные ошибки, возникающие при создании пользовательских панелей инструментов

Независимо от приложения и того, как оно хранит пользовательские панели, в VBA-программах нельзя выполнять следующие действия (попытка приводит к ошибке приложения):

- удалить панель, которая в этот момент не существует;
- создать новую панель, если ее имя совпадает с именем уже существующей панели;
- любым образом обратиться к элементам или свойствам несуществующей панели.

Конструктивный вывод из этих печальных фактов — до выполнения таких действий мы должны убедиться в наличии или отсутствии панели. Так как это требуется довольно часто, напишем библиотечную функцию, например, следующую.

```
Public Function CommandBarExists(CommandBarName As String) As Boolean
   Dim cb As CommandBar
   CommandBarExists = False
   For Each cb In CommandBars
       If cb.Name = CommandBarName Then
          CommandBarExists = True
          Exit For
      End If
   Next
End Function
```

Пункты меню со встроенными функциями и картинками

В пользовательское меню можно включать пункты из уже существующих меню со встроенными функциями. Например, если в меню должна быть команда **Сохранить**, почему бы ее не позаимствовать из панели инструментов **File**. Как это делается? Да, очень просто. Надо определить идентификатор этой команды свойством `ID`, а уже затем определенный пункт меню (в данном случае, **Сохранить**) добавить в строку меню методом `Add` семейства `Controls`, у которого в качестве значения параметра `ID` указано значение идентификатора данной команды.

Если же требуется в пункт меню вставить только картинку из стандартного пункта без инициализации ассоциированных с ним встроенных функций, то в этом случае по идентификатору команды методом `FindControl` семейства `CommandBars` надо получить ссылку на сам объект `CommandBarButton`, далее методом `CopyFace` объекта `CommandBarButton` скопировать рисунок найденной стандартной команды в буфер обмена, а затем уже методом `PasteFace` объекта `CommandBarButton` вставить его в искомую пользовательскую команду.

Для примера создадим строку меню, расположенную вдоль нижнего края окна приложения, и состоящую из выпадающего меню **Файл**, в которое входят пункты **Создать, Сохранить** и **Закрыть** (листинги 9.2, *а*, *б*). Пункты меню **Сохранить** и **Закрыть** позаимствуем из встроенного меню **File**. В пункт меню **Создать** вставим только значок из пункта **Создать** встроенного меню **File**, а его функциональность реализуем при помощи процедуры-заглушки (рис. 9.2).

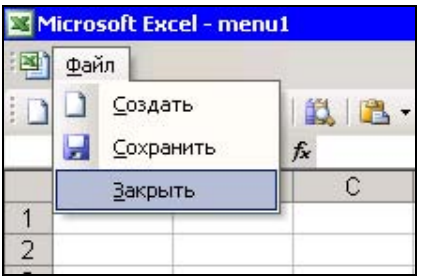


Рис. 9.2. Пункты меню со встроенными функциями и картинками

Листинг 9.2, *а*. Пункты меню со встроенными функциями и картинками. Модуль `ЭтаКнига`

```
Private Sub Workbook_Open()
   MenuBuilder
End Sub
```

```
Private Sub Workbook_BeforeClose(Cancel As Boolean)
  MenuKiller
End Sub
```

Листинг 9.2, *б*. Пункты меню со встроенными функциями и картинками. Стандартный модуль

```
Const CommandBarName As String = "Custom CommandBar"

Sub MenuBuilder()
   Dim cbMenuBar As CommandBar
   Dim cbMenu As CommandBarControl
   Dim btnNew As CommandBarButton
   Dim btnSave As CommandBarButton
   Dim btnExit As CommandBarButton
   Dim btnTemp As CommandBarButton
   Dim n As Integer
   Set cbMenuBar = Application.CommandBars.Add(Name:=CommandBarName, _
                                               Position:=msoBarTop, _
                                               MenuBar:=True, _
                                               Temporary:=True)
   cbMenuBar.Visible = True
   Set cbMenu = cbMenuBar.Controls.Add(Type:=msoControlPopup)
   cbMenu.Caption = "&Файл"
   Set btnNew = cbMenu.Controls.Add(Type:=msoControlButton)

   n = Application.CommandBars("File").Controls("Сохранить").ID
   Set btnSave = cbMenu.Controls.Add(Type:=msoControlButton, ID:=n)
   n = Application.CommandBars("File").Controls("Закрыть").ID
   Set btnExit = cbMenu.Controls.Add(Type:=msoControlButton, ID:=n)


   n = Application.CommandBars("File").Controls("Создать...").ID
   Set btnTemp = Application.CommandBars("File").FindControl( _
                                        Type:=msoControlButton, ID:=n)
   btnTemp.CopyFace
   With btnNew
```

```
      .PasteFace
      .Caption = "&Создать"
      .OnAction = "MacroNew"
   End With
End Sub

Public Sub MacroNew()
   MsgBox "Процедура-заглушка для команды Создать"
End Sub

Sub MenuKiller()
   On Error Resume Next
   Application.CommandBars(CommandBarName).Delete
   On Error GoTo 0
End Sub
```

Добавление нового элемента в существующее меню

VBA позволяет не только создавать полностью новое меню, но и добавлять в уже существующее новые элементы. Для того чтобы это сделать, достаточно определить местоположение того элемента, перед которым надо вставить новый, затем создать новый и поместить его в указанное место с помощью метода `Add`. В следующем примере (листинги 9.3, *а*, *б*) в документе Word в меню **Файл** после команды **Открыть** добавляется новая команда **Нажми меня** (рис. 9.3).

Рис. 9.3. Новая команда **Нажми меня** в меню **Файл**

Листинг 9.3, *а*. Добавление нового элемента в существующее меню File в документе Word. Модуль `ThisDocument`

```
Private Sub Document_Open()
   Dim cb As CommandBar
   Dim NewItem As CommandBarControl
   Dim oSaveAsMenu As CommandBarControl
   Dim ItemPosition As Integer
   Set cb = CommandBars.Item("File")
   Dim i As Integer
   For i = 1 To cb.Controls.Count
      If cb.Controls(i).Caption = "&Открыть..." Then
         ItemPosition = i + 1
         Exit For
      End If
   Next
   Set NewItem = cb.Controls.Add(Type:=msoControlButton, _
                              Before:=ItemPosition, _
                              Temporary:=True)
   With NewItem
      .Caption = "Нажми меня"
      .OnAction = "DoPressMe"
   End With
End Sub

Private Sub Document_Close()
   Dim cb As CommandBar
   Dim i As Integer
   Set cb = CommandBars.Item("File")
   Dim c As CommandBarControl
   For Each c In cb.Controls
      If c.Caption = "Нажми меня" Then
         c.Delete
         Exit For
      End If
   Next
End Sub
```

Листинг 9.3, *б*. Добавление нового элемента в существующее меню File в документе Word. Стандартный модуль

```
Sub DoPressMe()
   Msgbox "Test!"
End Sub
```

Создание панели инструментов

Панель инструментов является объектом `CommandBar`, который создается методом `Add` семейства `CommandBars`, при этом в качестве значения параметра `MenuBar` этого метода надо указать `False`.

```
Dim cb As CommandBar
Set cb = Application.CommandBars.Add(Name:=CommandBarName, _
                                     Position:=msoBarTop,
                                     MenuBar:=False, _
                                     Temporary:=True)
```

Элементами панели инструментов могут быть кнопки и раскрывающиеся списки. Продемонстрируем технику построения панели на простом примере. Панель будет иметь следующую структуру (рис. 9.4). Раскрывающийся список **Drop down menu**, состоящий из трех элементов (**MenuItem1**, **MenuItem2**, **Delete the User CommandBar**) и разделительной линии, а также пять кнопок (— кнопка со значком из встроенной кнопки, — кнопка со значком из внедренного в рабочий лист рисунка, Button3 — кнопка с надписью и со значком из внедренного в рабочий лист рисунка, Button4 — кнопка с надписью и со значком из файла и Button5 — кнопка с надписью). Выбор команды **Delete the User CommandBar** будет вызывать удаление созданной панели инструментов. С кнопками и другими командами раскрывающегося списка связана "процедура-заглушка", подтверждающая сделанный выбор.

В данном проекте код набирается в двух модулях: **ЭтаКнига** и стандартном.

В модуле **ЭтаКнига** (листинг 9.4, *а*) имеются две процедуры. Первая из них обрабатывает событие `Open` объекта `Workbook`. Эта процедура вызывает процедуру `CommandBarBuilder`, которая создает панель инструментов. Эта панель удаляется при закрытии книги. Для этого в коде обрабатывается событие `BeforeClose` объекта `Workbook`. В процедуре обработки этого события вызывается процедура `CommandBarKiller`, которая как раз и удаляет пользовательскую панель инструментов.

В модуле рабочего листа (листинг 9.4, *б*) объявлена постоянная `CommandBarName`, задающая имя панели инструментов, и определены процедуры

CommandBarBuilder и CommandBarKiller. Кроме того, в нем определены "процедура-заглушка" DoMacro и вспомогательная процедура CopyPictureFromFile, которая сначала вставляет рисунок из файла в рабочий лист, а затем — в буфер обмена. Всем кнопкам назначены всплывающие подсказки при помощи свойства TooltipText.

Для реализации данного проекта в книге должен существовать рабочий лист с именем **Значки,** в который внедрены два рисунка с именами CustomIcon1 и CustomIcon2.

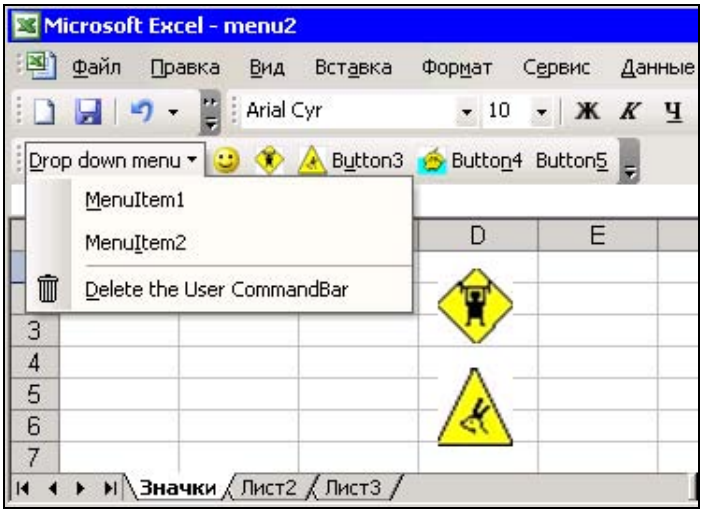


Рис. 9.4. Панель инструментов с раскрывающимся списком и пятью кнопками

Листинг 9.4, *а*. Панель инструментов. Модуль ЭтаКнига

```
Option Explicit

Private Sub Workbook_Open()
   CommandBarBuilder
End Sub

Private Sub Workbook_BeforeClose(Cancel As Boolean)
   CommandBarKiller
End Sub
```

Листинг 9.4, *б*. Панель инструментов. Стандартный модуль

```
Option Explicit

Const CommandBarName As String = "Custom CommandBar"

Sub CommandBarBuilder()
   Dim ws As Worksheet
   Set ws = Worksheets("Значки")
   Dim cb As CommandBar, cbDropDownMenu As CommandBarPopup
   Dim cbButton As CommandBarButton
   '  Удаление панели инструментов, если она существует
   CommandBarKiller

   Set cb = Application.CommandBars.Add(Name:=CommandBarName, _
                                        Position:=msoBarTop, _
                                        MenuBar:=False, _
                                        Temporary:=True)
   ' Добавление выпадающего меню
   Set cbDropDownMenu = cb.Controls.Add(Type:=msoControlPopup, _
                                        Temporary:=True)
   cbDropDownMenu.Caption = "&Drop down menu"

   ' Добавление пункта в выпадающее меню
   With cbDropDownMenu.Controls.Add(Type:=msoControlButton, _
                                    Temporary:=True)
      .Caption = "&MenuItem1"
      .Tag = "MenuItem1"
      .OnAction = "DoMacro"
   End With
   ' Добавление пункта в выпадающее меню
   With cbDropDownMenu.Controls.Add(Type:=msoControlButton, _
                                    Temporary:=True)
      .Caption = "Menu&Item2"
      .Tag = "MenuItem2"
      .OnAction = "DoMacro"
   End With
   ' Добавление пункта в выпадающее меню
```

```
With cbDropDownMenu.Controls.Add(Type:=msoControlButton, _
                                 Temporary:=True)
   .Caption = "&Delete the User CommandBar"
   .OnAction = "CommandBarKiller"
   .FaceId = 67 ' ID корзины
   .Style = msoButtonIconAndCaption
   .BeginGroup = True
End With

' Добавление кнопки с улыбающимся лицом
Set cbButton = cb.Controls.Add(Type:=msoControlButton, _
                               Temporary:=True)
With cbButton
   .Caption = "&Button1"
   .FaceId = 59 ' ID улыбающегося лица
   .Style = msoButtonIcon
   .Tag = "Button1"
   .OnAction = "DoMacro"
   .TooltipText = "Кнопка со встроенным рисунком"
End With

' Добавление кнопки со значком, скопированным с листа
Set cbButton = cb.Controls.Add(Type:=msoControlButton, _
                               Temporary:=True)
With cbButton
   .Caption = "Butt&on2"
   .Tag = "Button2"
   .Style = msoButtonIcon
   .OnAction = "DoMacro"
   ws.Shapes("CustomIcon1").Copy
   .PasteFace ' Вставка пользовательского значка
   .TooltipText = "The button with a custom icon"
End With

' Добавление кнопки со значком, скопированным с листа
Set cbButton = cb.Controls.Add(Type:=msoControlButton, _
                               Temporary:=True)
With cbButton
```

```
    .Caption = "B&utton3"
    .Tag = "Button3"
    .Style = msoButtonIconAndCaption
    .OnAction = "DoMacro"
    ws.Shapes("CustomIcon2").Copy
    .PasteFace ' Вставка пользовательского значка
    .TooltipText = "Кнопка с пользовательским значком"
End With

' Добавление кнопки со значком из файла
Set cbButton = cb.Controls.Add(Type:=msoControlButton, _
                               Temporary:=True)
With cbButton
    .Caption = "Butto&n4"
    .Tag = "Button4"
    .Style = msoButtonIconAndCaption
    .OnAction = "DoMacro"
' Вставка значка из файла
    If CopyPictureFromFile(ws, ThisWorkbook.Path & "\hippo.jpg") Then
       .PasteFace
    End If
    .TooltipText = "Кнопка со значком из файла"
End With

' Добавление кнопки с текстом
Set cbButton = cb.Controls.Add(Type:=msoControlButton, _
                               Temporary:=True)
With cbButton
    .Caption = "Button&5"
    .Tag = "Button5"
    .OnAction = "DoMacro"
    .Style = msoButtonCaption
    .TooltipText = "Это просто кнопка"
End With

cb.Visible = True
Set cbButton = Nothing
Set cbDropDownMenu = Nothing
```

```
   Set cb = Nothing
End Sub

Sub CommandBarKiller()
   On Error Resume Next
   Application.CommandBars(CommandBarName).Delete
   On Error GoTo 0
End Sub

Sub DoMacro()
   If Application.CommandBars.ActionControl Is Nothing Then
      MsgBox "Нет ActionControl!"
      Exit Sub
   End If
   Dim text As String
   text = Application.CommandBars.ActionControl.Tag
   If text = "MenuItem1" Then
      MsgBox "Это MenuItem1!"
   ElseIf text = "MenuItem2" Then
      MsgBox "Это MenuItem2!"
   ElseIf text = "Button1" Then
      MsgBox "Это Button1!"
   ElseIf text = "Button2" Then
      MsgBox "Это Button2!"
   ElseIf text = "Button3" Then
      MsgBox "Это Button3!"
   ElseIf text = "Button4" Then
      MsgBox "Это Button4!"
   ElseIf text = "Button5" Then
      MsgBox "Это Button5!"
   End If
End Sub

Function CopyPictureFromFile(ws As Worksheet, _
                             SourceFile As String) As Boolean
   If Len(Dir(SourceFile)) = 0 Then
      MsgBox "No file " & SourceFile
      CopyPictureFromFile = False
```

```
      Exit Function
   End If
   Dim pic As Object
   On Error GoTo NoPicture
   Set pic = ws.Pictures.Insert(SourceFile)
   pic.CopyPicture xlScreen, xlPicture
   pic.Delete
   Set pic = Nothing
   CopyPictureFromFile = True
   Exit Function
NoPicture:
   MsgBox "Something is wrong with " & SourceFile
   CopyPictureFromFile = False
End Function
```

Расположение нескольких панелей инструментов в один ряд

В один ряд можно расположить несколько панелей инструментов (рис. 9.5). Для этого надо воспользоваться свойством `RowIndex` объекта `CommandBar`, который и задает их порядок, как, например, показано в листинге 9.5. Допустимыми значениями этого свойства могут быть либо целые положительные константы, либо константы `MsoBarRow`: `msoBarRowFirst` и `msoBarRowLast`.

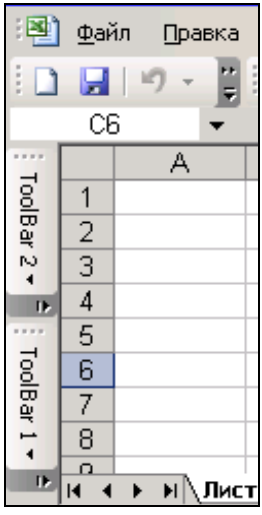


Рис. 9.5. Две панели инструментов в один ряд

Листинг 9.5. Расположение двух панелей инструментов в один ряд

```
Sub SetTwoToolBars()
   Dim cbMenu As CommandBarControl
   With Application.CommandBars
      With .Add("My ToolBar 1")
         .Visible = True
         .Position = msoBarLeft
         .RowIndex = 1
         Set cbMenu = .Controls.Add(Type:=msoControlPopup)
         cbMenu.Caption = "ToolBar 1"
      End With
      Set cbMenu = Nothing
      With .Add("My ToolBar 2")
        .Visible = True
        .Position = msoBarLeft
        .RowIndex = 1
        Set cbMenu = .Controls.Add(Type:=msoControlPopup)
        cbMenu.Caption = "ToolBar 2"
      End With
   End With
End Sub
```

Добавление раскрывающегося списка в панель инструментов

Элементами панели инструментов могут быть не только кнопки, но и раскрывающиеся списки, которые являются объектами `CommandBarComboBox`. Заполняются эти списки методом `AddItem`. Свойство `DropDownLines` задает число отображаемых в нем элементов. Свойство `ListIndex` — индекс выбранного элемента. Для того чтобы обработать событие `Change` объектами `CommandBarComboBox`, необходимо этот объект объявить с помощью ключевого слова `WithEvents`. В следующем примере (листинги 9.6, *а*, *б*) конструируется меню, в котором имеется один раскрывающий список, состоящий из трех элементов **Red**, **Yellow** и **Green** (рис. 9.6). Выбор соответствующего элемента приводит к отображению окна с подтверждением и окраской активного листа в указанный цвет.

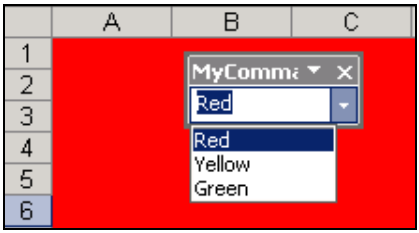


Рис. 9.6. Раскрывающийся список в панель инструментов

Листинг 9.6, *а*. Добавление раскрывающегося списка в панель инструментов. Модуль ЭтаКнига

```
Private ctlComboBoxHandler As New ComboBoxHandler

Private Sub Workbook_Open()
   AddComboBox
End Sub

Sub AddComboBox()
   Set App = Application
   Dim cb As Office.CommandBar
   Set cb = App.CommandBars.Add(Name:="MyCommandBar", _
                                Temporary:=True)
   Dim cmb As Office.CommandBarComboBox
   Set cmb = cb.Controls.Add(Type:=msoControlComboBox)
   With cmb
      .AddItem "Red", 1
      .AddItem "Yellow", 2
      .AddItem "Green", 3
      .DropDownLines = 4
      .DropDownWidth = 75
      .ListHeaderCount = 0
      .ListIndex = 1
   End With
   ctlComboBoxHandler.SyncBox cmb
   cb.Visible = True
End Sub

Private Sub Workbook_BeforeClose(Cancel As Boolean)
   Application.CommandBars("MyCommandBar").Delete
End Sub
```

Листинг 9.6, *б*. Добавление раскрывающегося списка в панель инструментов. Модуль класса `ComboBoxHandler`

```
Private WithEvents ComboBoxEvent As Office.CommandBarComboBox

Public Sub SyncBox(box As Office.CommandBarComboBox)
   Set ComboBoxEvent = box
   If Not box Is Nothing Then
      MsgBox "Synced " & box.Caption & " ComboBox events."
   End If
End Sub

Private Sub Class_Terminate()
   Set ComboBoxEvent = Nothing
End Sub

Private Sub ComboBoxEvent_Change(ByVal Ctrl As Office.CommandBarComboBox)
   Dim stComboText As String
   stComboText = Ctrl.Text
   Select Case stComboText
      Case "Red"
         DoRed
      Case "Yellow"
         DoYellow
      Case "Green"
         DoGreen
   End Select
End Sub

Private Sub DoRed()
   MsgBox "Red"
   ActiveSheet.Cells.Interior.Color = RGB(255, 0, 0)
End Sub

Private Sub DoYellow()
   MsgBox "Yellow"
   ActiveSheet.Cells.Interior.Color = RGB(255, 255, 0)
```

```
End Sub

Private Sub DoGreen()
   MsgBox "Green"
   ActiveSheet.Cells.Interior.Color = RGB(0, 255, 0)
End Sub
```

Конструирование контекстного меню

Методы создания строк меню и их элементов управления можно применить к разработке контекстного меню. Для построения контекстного меню надо:

1. Создать контекстное меню.
2. Добавить в контекстное меню элементы управления и связать с ними макросы.
3. Задать момент вывода контекстного меню.
4. Позаботится о том, чтобы после вывода пользовательского контекстного меню не отображалось встроенное в Excel контекстное меню.

Первые два пункта намеченной программы реализуются точно так же, как и при построении обычного меню. Созданное контекстное меню отображается на экране методом `ShowPopup` объекта `CommandBar`. Для задания момента отображения контекстного меню лучше всего подходит процедура обработки события `SheetBeforeRightClick` объекта `Workbook`. Параметры этой процедуры `Sh` и `Target` позволяют локализовать место в рабочей книге, в котором это меню должно отображаться, а именно, рабочий лист и диапазон. Если необходимо отображение контекстного меню при нажатии правой кнопки мыши в любом месте любого рабочего листа книги, то о значениях параметров `Sh`

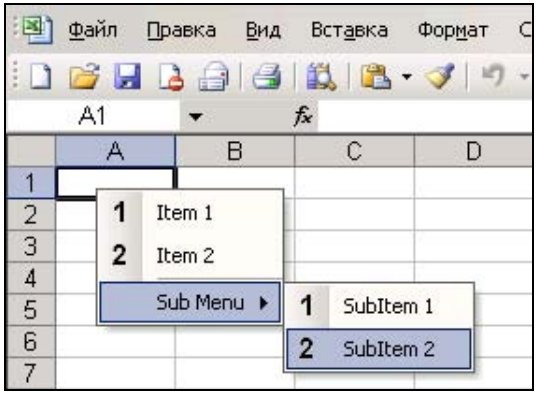


Рис. 9.7. Контекстное меню

и `Target` нет нужды беспокоиться. Установив значение параметра `Cancel` процедуры обработки события `SheetBeforeRightClick` объекта `Workbook` равным `True`, можно отключить выполнения тех функций, которые по умолчанию связаны с нажатием правой кнопки мыши.

Следующий код (листинги 9.7, *а*, *б*) демонстрирует технику создания контекстного меню на примере меню, состоящего из двух элементов и выпадающего меню, имеющего также два элемента. Отображение пользовательского контекстного меню произойдет при щелчке правой кнопкой на ячейке **A1** (рис. 9.7).

Листинг 9.7, *а*. Контекстное меню. Модуль рабочего листа

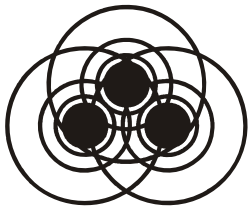
```
Private Sub Workbook_Open()
   BuilderPopUpMenu
End Sub

Private Sub Workbook_BeforeClose(Cancel As Boolean)
   Application.CommandBars("TemporaryPopupMenu").Delete
End Sub

Private Sub Workbook_SheetBeforeRightClick(ByVal Sh As Object, _
                                           ByVal Target As Range, _
                                           Cancel As Boolean)
   If Not (Application.Intersect(Target, Range("A1")) Is Nothing) Then
      Application.CommandBars("TemporaryPopupMenu").ShowPopup
      Cancel = True
   End If
End Sub
```

Листинг 9.7, *б*. Контекстное меню. Стандартный модуль

```
Sub BuilderPopUpMenu()
   Dim cb As CommandBar
   Dim cbPopUp As CommandBarPopup
   Set cb = Application.CommandBars.Add( _
                        Name:="TemporaryPopupMenu", _
                        Position:=msoBarPopup)
   With cb
      With .Controls.Add(Type:=msoControlButton)
         .OnAction = "MyMacroName"
         .FaceId = 71
```

```
            .Caption = "Item 1"
            .Tag = "Item 1"
            .TooltipText = "Tooltip For Item 1"
        End With
        With .Controls.Add(Type:=msoControlButton)
            .OnAction = "MyMacroName"
            .FaceId = 72
            .Caption = "Item 2"
            .Tag = "Item 2"
            .TooltipText = "Tooltip For Item 2"
        End With
        Set cbPopUp = .Controls.Add(Type:=msoControlPopup)
        With cbPopUp
            .BeginGroup = True
            .Caption = "Sub Menu"
            With .Controls.Add(Type:=msoControlButton)
                .OnAction = "MyMacroName"
                .FaceId = 71
                .Caption = "SubItem 1"
                .Tag = "SubItem 1"
                .TooltipText = "Tooltip For SubItem 1"
            End With
            With .Controls.Add(Type:=msoControlButton)
                .OnAction = "MyMacroName"
                .FaceId = 72
                .Caption = "SubItem 2"
                .Tag = "SubItem 2"
                .TooltipText = "Tooltip For SubItem 2"
            End With
        End With
    End With
    Set cbPopUp = Nothing
    Set cb = Nothing
End Sub

Sub KillerPopUpMenu()
    On Error Resume Next
    CommandBars("TemporaryPopupMenu").Delete
    On Error GoTo 0
End Sub
```

Глава 10



Диаграммы

MS Excel обладает мощным средством — мастером диаграмм, позволяющим создавать различные диаграммы в наглядном виде. VBA предоставляет средства как для автоматизации процесса конструирования диаграмм, так и для управления ими, добавляя в проекты итеративность и компактность. В этой главе вы узнаете, как конструируется диаграмма, как получается ссылка на активную диаграмму, как изменить параметры диаграммы методом `ChartWizard`, как построить внедренную диаграмму, как установить размеры и местоположение внедренной диаграммы, как сконструировать диаграмму на основе массива данных, как сохранить диаграмму в виде графического файла, как установить цвета серий, как отобразить рисунок вместо заливки серий, как специфицировать градиентную заливку серий, как изменять параметры диаграммы, как печатать и осуществлять предварительный просмотр печати диаграммы, как строить диаграмму на основе данных из нескольких листов, как защищать диаграмму, как изменять диапазон, на основе которого конструируется диаграмма, как создавать анимацию, как создавать контекстное меню для диаграммы.

Построение диаграммы

Для того чтобы построить диаграмму, достаточно воспользоваться методом `Add Charts`, у которого имеются три необязательных параметра: *`Before`* — указывает на лист, перед которым вставляется диаграмма, *`After`* — задает ссылку на лист, после которого вставляется диаграмма, *`Count`* — специфицирует число вставляемых листов.

```
Add(Before, After, Count)
```

Кроме того, надо указать ссылку на тот диапазон, на основе которого строится диаграмма. Здесь на помощь приходит метод `SetSourceData`, параметр `Source` которого и дает искомую ссылку. Далее, используя большую коллекцию свойств диаграммы, можно задать ее внешний вид. Например, свойство `ChartTitle` реализует доступ к ее заголовку, данные о котором инкапсулированы в объекте `ChartTitle`. В частности, его свойство `Text` позволяет

задавать текст заголовка. Но прежде чем работать с заголовком, надо установить значение свойства `HasTitle` объекта `Chart` равным `True`. Следующий код (листинг 10.1) демонстрирует то, как на основе табличных данных, размещенных в диапазоне **A1:E4** можно создать диаграмму (рис. 10.1).

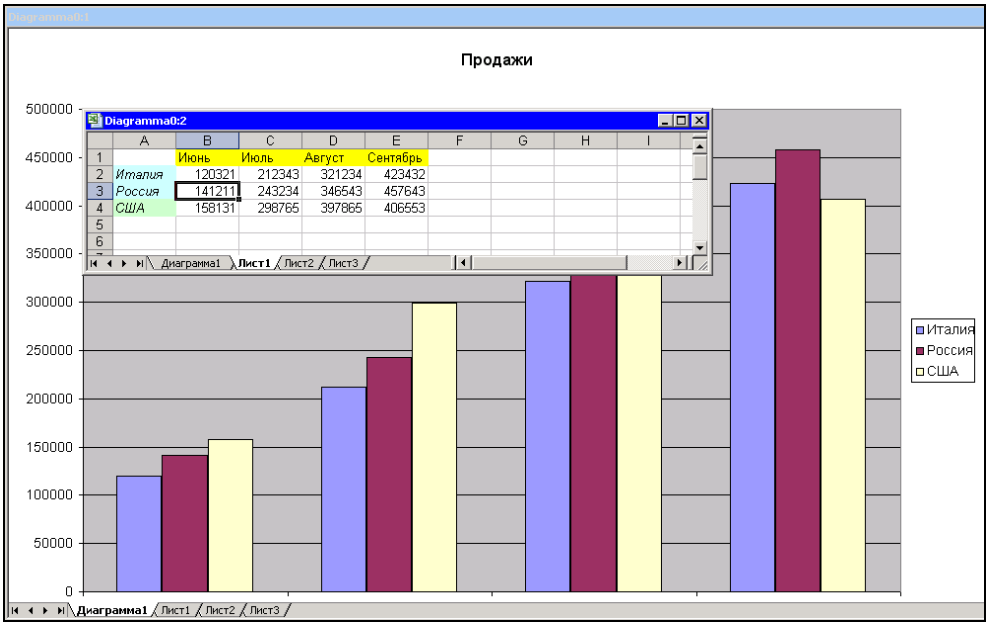


Рис. 10.1. Построение диаграммы

Листинг 10.1. Построение диаграммы

```
Sub CreateChart()
    Dim c As Chart
    Set c = Charts.Add
    c.SetSourceData Source:=Worksheets(1).Range("A1:E4")
    c.HasTitle = True
    c.ChartTitle.Text = "Продажи"
    c.Activate
End Sub
```

Как получить ссылку на активную диаграмму

Свойство `ActiveChart` объектов `Application`, `Window` и `Workbook` возвращает ссылку на активную диаграмму. Если таковой нет, то оно возвращает значение `Nothing`.

Как методом *ChartWizard* изменить параметры диаграммы

Метод `ChartWizard` позволяет модифицировать диаграмму, без установки ее свойств по раздельности, т. е. он предоставляет возможность управлять диаграммой в целом.

`ChartWizard(`*`Source`*`, `*`Gallery`*`, `*`Format`*`, `*`PlotBy`*`, `*`CategoryLabels`*`, ↳`*`SeriesLabels`*`, `*`HasLegend`*`, `*`Title`*`, `*`CategoryTitle`*`, `*`ValueTitle`*`, `*`ExtraTitle`*`)`, где:

- *`Source`* — задает диапазон с данными, на основе которого строится диаграмма;
- *`Gallery`* — устанавливает тип диаграммы. Допустимы значения: `xlArea`, `xlBar`, `xlColumn`, `xlLine`, `xlPie`, `xlRadar`, `xlXYScatter`, `xlCombination`, `xl3DArea`, `xl3DBar`, `xl3DColumn`, `xl3DLine`, `xl3Dpie`, `xl3Dsurface`, `xlDoughnut` или `xlDefaultAutoFormat`;
- *`Format`* — определяет формат для данного типа диаграммы. Каждый тип имеет до десяти форматов;
- *`PlotBy`* — задает, как располагаются данные в диапазоне, на основе которого строится диаграмма. Допустимые значения: `xlRows` (по строкам) и `xlColumns` (по столбцам);
- *`CategoryLabels`* — определяет число строк (столбцов) с метками категорий (т. е. данных отводимых под ось абсцисс);
- *`SeriesLabels`* — устанавливает число строк (столбцов) с метками рядов (т. е. данных, отводимых под заголовки);
- *`HasLegend`* — специфицирует, надо ли отображать легенду;
- *`Title`* — задает заголовок диаграммы;
- *`CategoryTitle`* — определяет название оси категорий;
- *`ValueTitle`* — устанавливает название оси значений;
- *`ExtraTitle`* — специфицирует название оси серий для трехмерной диаграммы или название второй оси значений для двухмерной диаграммы.

Изменение типа диаграммы с помощью метода *ChartWizard*

В качестве примера создадим приложение, изменяющее тип диаграммы. В рабочей книге имеются рабочий лист **Итого** и лист диаграмм **Диаграмма**. На листе **Итого** в диапазоне **A1:E4** приведены результаты продаж некоторой фирмы за отчетный период, а на листе **Диаграмма** — диаграмма, сконструированная на основе данных из диапазона **A1:E4**. Кроме того, на рабочем

листе **Итого** расположен список, у которого при помощи окна **Properties** установлено значение свойства `Name` равным `lstType`. В список выводятся названия типов диаграмм, причем выбор элемента приводит к изменению типа диаграммы (рис. 10.2).

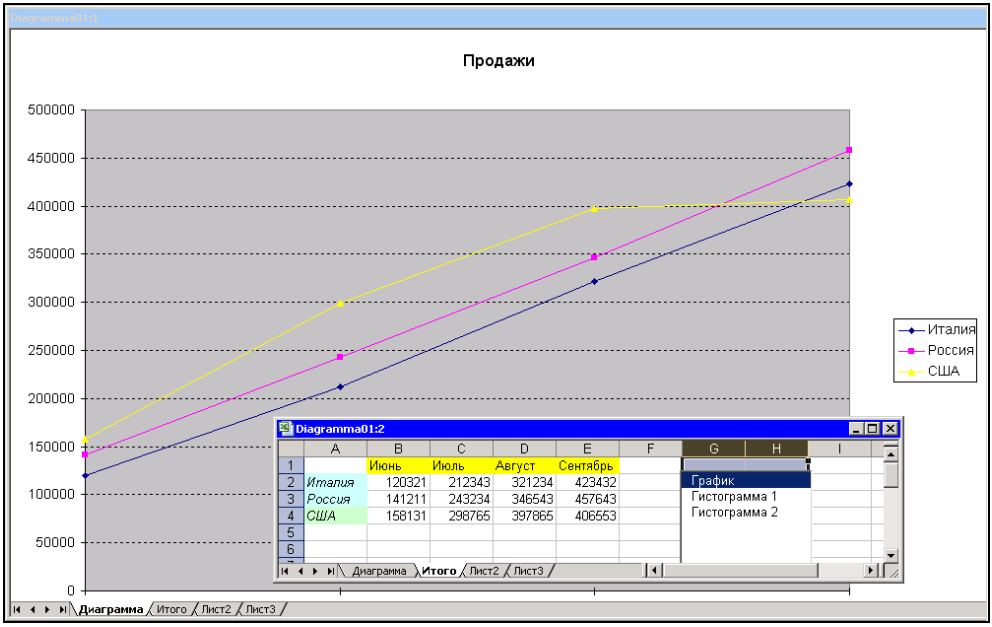


Рис. 10.2. Изменение типа диаграммы

Для реализации этого проекта в модуле **ЭтаКнига** надо набрать код из листинга 10.2, *а*, в котором заполнение списка реализуется процедурой `FilllstType`, вызов которой осуществляется в процедуре обработки события `Open` объекта `Workbook`.

Листинг 10.2, *а*. Изменение типа диаграммы. Модуль ЭтаКнига

```
Private Sub Workbook_Open()
   FilllstType
End Sub

Private Sub FilllstType()
   Dim tb(2, 1) As Variant
   tb(0, 0) = "График": tb(0, 1) = xlLine
   tb(1, 0) = "Гистограмма 1": tb(1, 1) = xlBar
```

```
   tb(2, 0) = "Гистограмма 2": tb(2, 1) = xlColumn
   With Worksheets("Итого").lstType
      .Clear
      .ColumnCount = 2
      .TextColumn = 2
      .ColumnWidths = "70;0"
      .List = tb
      .ListIndex = 0
   End With
End Sub
```

Смена типа диаграммы реализуется в процедуре обработки `Click` списка из листинга 10.2, *б*, в которой значение параметра `Gallery` метода `ChartWizard`, примененного к данной диаграмме, устанавливается равным выбранному из списка значению.

Листинг 10.2, *б*. Изменение типа диаграммы. Модуль рабочего листа

```
Private Sub lstType_Click()
   Charts("Диаграмма").ChartWizard Gallery:=lstType.Text
End Sub
```

Построение внедренной диаграммы

Внедренные диаграммы являются объектами `ChartObject` и в своей совокупности образуют семейство `ChartObjects`. В методе `Add` этого семейства параметры *`Left`*, *`Top`*, *`Width`*, *`Height`* определяют местоположение верхнего левого угла диаграммы и ее размеры.

`Add(`*`Left`*`, `*`Top`*`, `*`Width`*`, `*`Height`*`)`

Следующий код (листинг 10.3) как раз демонстрирует то, как на основе данных из раздела *Построение диаграммы* конструируется внедренная диаграмма (рис. 10.3). Как видите, техника в обоих случаях одна и та же.

Листинг 10.3. Внедренная диаграмма

```
Sub CreateChartObject()
   Dim c As ChartObject
   Set c = Worksheets(1).ChartObjects.Add(100, 60, 250, 200)
   c.Chart.ChartType = xl3DArea
   c.Chart.SetSourceData Source:=Worksheets(1).Range("A1:E4")
End Sub
```

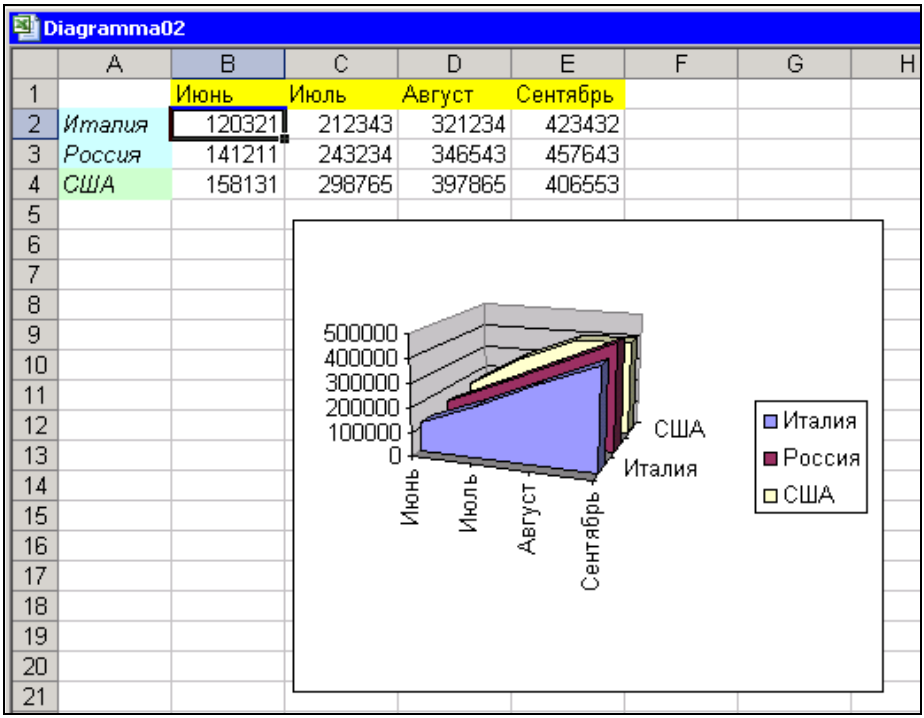


Рис. 10.3. Внедренная диаграмма

Установка размеров и местоположения внедренной диаграммы

Размеры диаграммы можно установить на этапе ее создания при помощи метода `Add`. А как быть, если диаграмма уже создана? Здесь на помощь приходят свойства `Left`, `Top`, `Width`, `Height`. В следующем коде (листинг 10.4), например, они применяются для расположения активной внедренной диаграммы поверх диапазона **A5:G17**.

Листинг 10.4. Установка размеров и местоположения внедренной диаграммы

```
Sub LocateActiveChart()
   If ActiveChart Is Nothing Then Exit Sub
   ActiveChart.Parent.Top = Range("A5").Top
   ActiveChart.Parent.Left = Range("A5").Left
   ActiveChart.Parent.Width = Range("A5:G17").Width
   ActiveChart.Parent.Height = Range("A5:G17").Height
End Sub
```

Построение диаграммы на основе массива данных

Диаграмму можно конструировать не только на основе данных из рабочего листа, но и на базе массива данных. То, как это можно сделать, демонстрируется в следующем коде (листинг 10.5).

Листинг 10.5. Конструирование диаграммы на основе массива данных

```
Sub ArrayChartBuilder()
   ActiveSheet.ChartObjects.Add(10, 10, 200, 200).Select
   ActiveChart.SetSourceData Cells(1, 1) 'Временная ссылка
   ActiveChart.SeriesCollection(1).Values = _
         Array(121, 123, 140, 169, 187, 123)
End Sub
```

Удаление диаграммы

Для удаления диаграммы или внедренной диаграммы ее надо идентифицировать. Это можно сделать либо по имени, либо по индексу, а далее остается применить метод `Delete`. Например, в следующем коде удаляется диаграмма с именем `Report`. Имя диаграммы устанавливается свойством `Name`.

```
Sub DeleteChart()
   On Error Resume Next
   Charts("Report").Delete
End Sub
```

Можно удалять не одну специфицированную диаграмму, а все семейство, применив к нему метод `Delete`. В следующем примере демонстрируется подобный подход. На рабочем листе **Италия** в диапазоне **B1:E2** собраны данные о продажах некоторой фирмы. В ячейках **A1** и **A2** приведены названия оси абсцисс и ординат конструируемой диаграммы. Заголовок диаграммы будет совпадать с названием листа. На диаграмме также будут отображаться таблица значений и сетка координат. Построение и удаление диаграммы будут производиться автоматически по нажатию кнопок **Построить диаграмму** и **Удалить диаграмму** (рис. 10.4).

Для реализации этого проекта на рабочем листе расположите две кнопки и, используя окно **Properties,** установите значение их свойств `Name` равными `btnBuilder` и `btnKiller`, а свойств `Caption` равными **Построить диаграмму** и **Удалить диаграмму**. В модуле рабочего листа набрать следующий код (листинг 10.6). Проект готов.

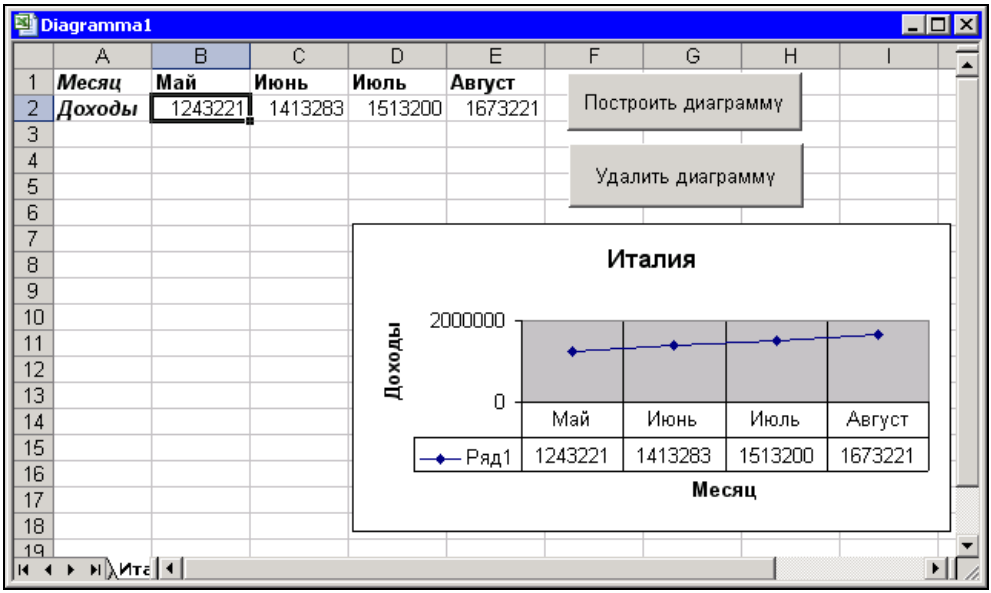


Рис. 10.4. Построение и удаление диаграммы

Листинг 10.6. Построение и удаление диаграммы

```
Const sname As String = "Италия"

Private Sub btnBuilder_Click()
   ChartBuilder
   Range("B2").Select
End Sub

Private Sub btnKiller_Click()
   ChartKiller
End Sub

Sub ChartBuilder()
   Charts.Add
   Dim rgn As Range
   Set rgn = Sheets(sname).Range("B1:E2")
   ActiveChart.ChartType = xlLineMarkers
   ActiveChart.SetSourceData Source:=rgn, PlotBy:=xlRows
   ActiveChart.Location Where:=xlLocationAsObject, Name:=sname
```

```
   With ActiveChart
      .HasTitle = True
      .ChartTitle.Characters.Text = sname
      .Axes(xlCategory, xlPrimary).HasTitle = True
      .Axes(xlCategory, xlPrimary).AxisTitle.Characters.Text _
            = Sheets(sname).Range("A1").Value
      .Axes(xlValue, xlPrimary).HasTitle = True
      .Axes(xlValue, xlPrimary).AxisTitle.Characters.Text _
            = Sheets(sname).Range("A2").Value
   End With
   With ActiveChart.Axes(xlCategory)
      .HasMajorGridlines = True
      .HasMinorGridlines = False
   End With
   With ActiveChart.Axes(xlValue)
      .HasMajorGridlines = True
      .HasMinorGridlines = False
   End With
   ActiveChart.HasLegend = False
   ActiveChart.HasDataTable = True
   ActiveChart.DataTable.ShowLegendKey = True
End Sub

Sub ChartKiller()
   ActiveSheet.ChartObjects.Delete
End Sub
```

Сохранение диаграммы в виде графического файла

Для сохранения диаграммы в виде графического файла надо воспользоваться методом `Export` объекта `Chart`.

`Export(`*`FileName`*`, `*`FilterName`*`, `*`Interactive`*`)`, где:

- *`FileName`* — обязательный параметр, задающий имя файла;
- *`FilterName`* — необязательный параметр, устанавливающий фильтр сохраняемых файлов;
- *`Interactive`* — необязательный параметр, устанавливающий, надо ли отображать диалоговое окно со спецификацией фильтра.

В пример из раздела *Удаление диаграммы* добавьте кнопку (рис. 10.5), у которой установите значения свойств `Name` и `Caption` равными `cmdSaveChart` и `Сохранить диаграмму`. При нажатии на эту кнопку будет отображаться диалоговое окно **Сохранение документа**, которое предоставит интерфейс для сохранения диаграммы в файл формата GIF (листинг 10.7).

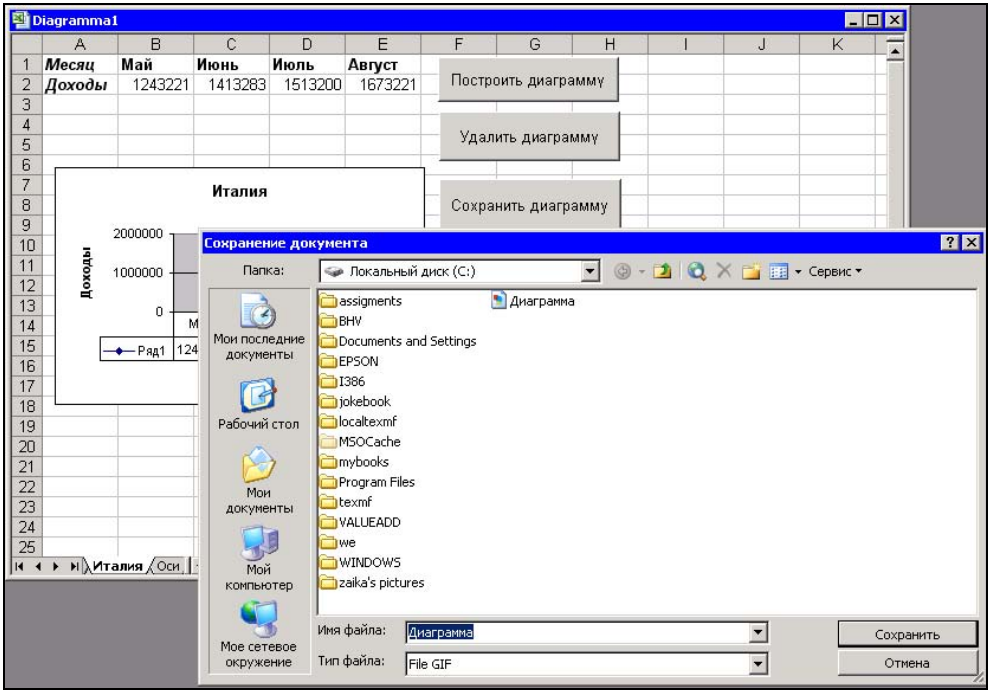


Рис. 10.5. Сохранение диаграммы как графического файла

Листинг 10.7. Сохранение диаграммы в виде графического файла

```
Private Sub cmdSaveChart_Click()
   Dim FileName As Variant
   FileName = Application.GetSaveAsFilename( _
         InitialFileName:=Charts(1).Name & ".gif", _
         FileFilter:="File GIF (*.gif), *.gif")
   If FileName <> False Then
       If ChartObjects.Count > 0 Then
          ChartObjects(1).Chart.Export FileName, "GIF"
       End If
   End If
End Sub
```

Задание цветов серий

При построении диаграмм можно автоматизировать процесс установки цветов серий данных. Например, на рабочем листе в диапазоне **A1:E4** имеется таблица с результатом работы фирмы за отчетный период. Для наглядности заголовки строк с названиями стран, в которые фирмой поставлялись товары, окрашены в различные цвета. Задача состоит в том, чтобы и в графике соответствующие серии были окрашены в те же цвета (рис. 10.6). Как это сделать? Очень просто. Добавьте на форму кнопку, при помощи окна **Properties** установите у нее в качестве значений свойств `Name` и `Caption` `Установить цвета` и `cmdSetColor`, а в модуле рабочего листа наберите следующий код (листинг 10.8).

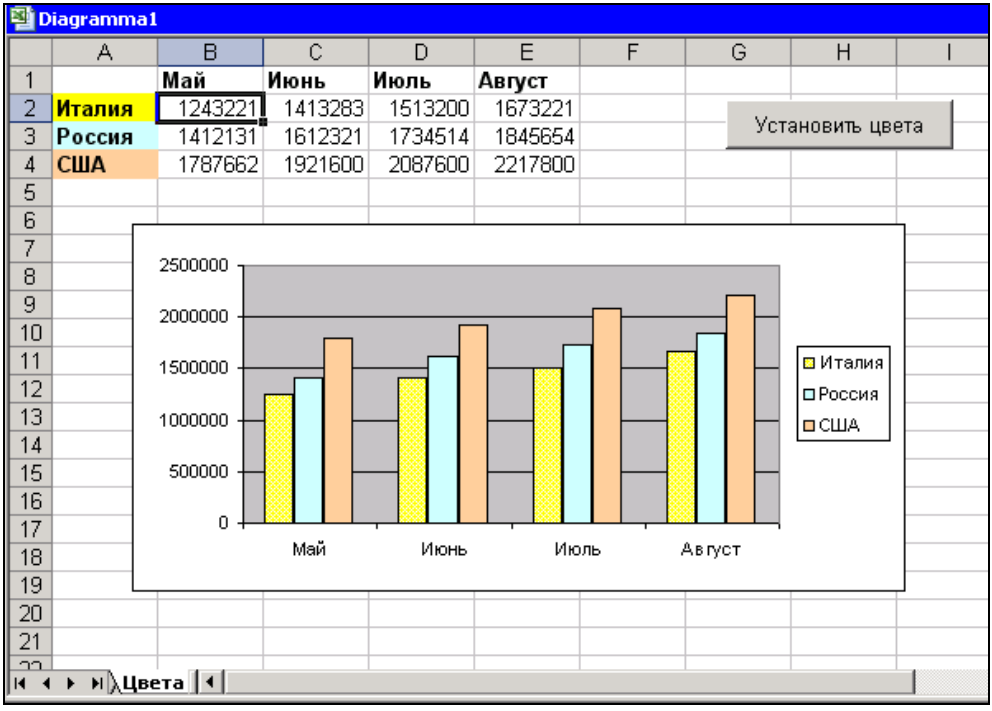


Рис. 10.6. Задание цветов серий

Листинг 10.8. Задание цветов серий

```
Private Sub cmdSetColor_Click()
   ActiveSheet.ChartObjects(1).Activate
   With ActiveChart
```

```
        .SeriesCollection(1).Interior.Color = Range("a2").Interior.Color
        .SeriesCollection(2).Interior.Color = Range("a3").Interior.Color
        .SeriesCollection(3).Interior.Color = Range("a4").Interior.Color
    End With
End Sub
```

Рисунок вместо заливки серий

Серии диаграмм можно не только закрашивать, но и вставлять в них рисунки. Как это сделать? Очень просто. Сначала надо запрограммировать вставку рисунка в рабочий лист. Это можно сделать методом `Insert` объекта `Picture`. Затем его выделить и скопировать в буфер обмена методом `Copy`. После чего остается только выделить искомую серию и вставить в нее методом `Paste` картинку из буфера. Следующее приложение демонстрирует такой подход (листинг 10.9). В нем дополнительно на рабочем листе расположена кнопка **Вставить рисунки** (рис. 10.7), у которой значение свойства `Name` установлено равным `cmdInputPicture`, нажатие на которую приводит к загрузке изображений растровых файлов с флагами соответствующих стран, размещению этих рисунков в строках с одноименными странами и вставке рисунков в соответствующие серии.

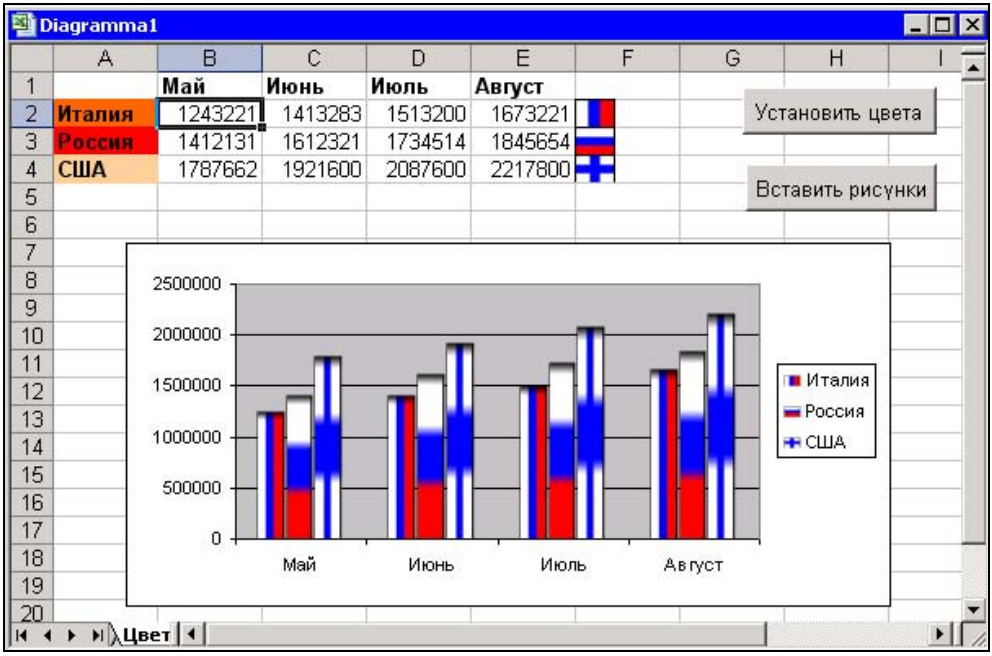


Рис. 10.7. Рисунок вместо заливки серий

Листинг 10.9. Рисунок вместо заливки серий

```
Private Sub cmdInputPicture_Click()
   InputPictures 6
End Sub

Private Sub InputPictures(col As Integer)
    On Error Resume Next
    Dim fnames
    fnames = Array("D:\france.bmp", "D:\russia.bmp", "D:\finland.bmp")
    Dim i As Integer
    For i = 0 To UBound(fnames)
       ActiveSheet.Pictures.Insert(fnames(i)).Select
       Selection.ShapeRange.Width = Cells(i + 2, col).Width
       Selection.ShapeRange.Height = Cells(i + 2, col).Height
       Selection.ShapeRange.Top = Cells(i + 2, col).Top
       Selection.ShapeRange.Left = Cells(i + 2, col).Left
       Selection.Copy
       ActiveSheet.ChartObjects(1).Activate
       ActiveChart.SeriesCollection(i + 1).Select
       Selection.Paste
       Cells(i + 2, col).Select
    Next
End Sub
```

Установка градиентной заливки серий

У графика можно задавать не только сплошную, но и градиентную заливку серий. Как это делается, продемонстрируем на примере из раздела *Задание цветов серий*, но теперь для установки цвета будем использовать две ячейки: **A2** и **F2** — для первой, **A3** и **F3** — для второй и **A4** и **F4** — для третьей серии (рис. 10.8). Значениями свойств `Caption` и `Name` в данном случае являются `Установить градиентную заливку` и `cmdSetGradientColor`. В модуле рабочего листа надо набрать следующий код (листинг 10.10).

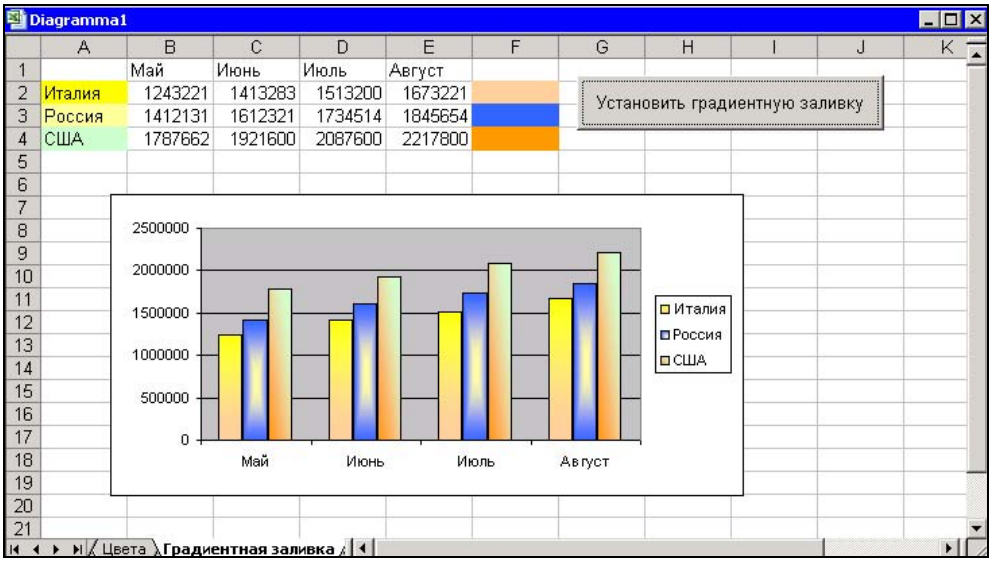


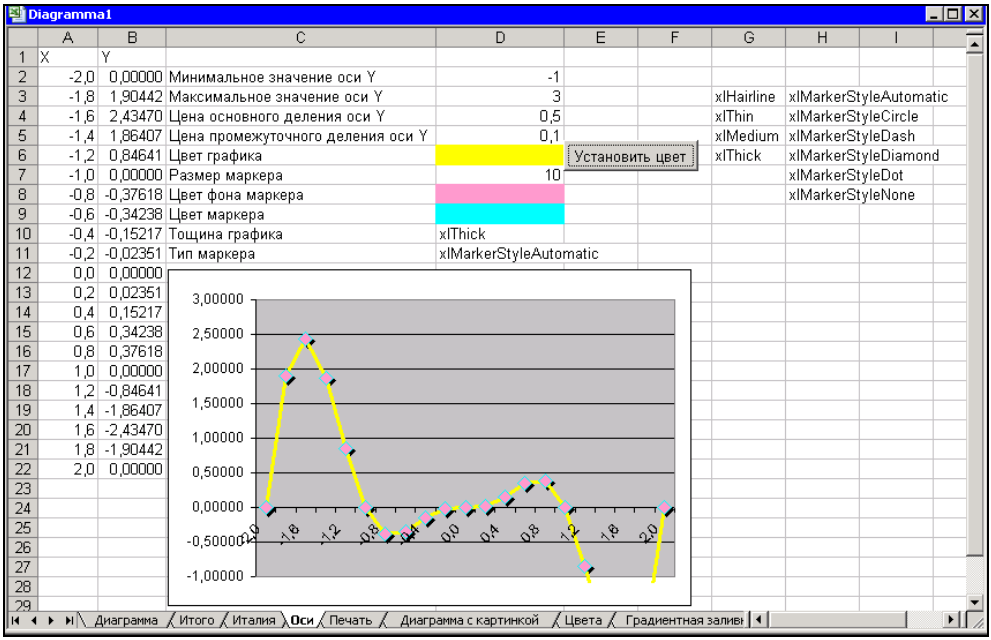
Рис. 10.8. Установка градиентной заливки серий

Листинг 10.10. Установка градиентной заливки серий

```
Private Sub cmdSetGradientColor_Click()
   ActiveSheet.ChartObjects(1).Activate
   ActiveChart.SeriesCollection(1).Select
   With Selection.Fill
     .ForeColor.SchemeColor = Range("A2").Interior.ColorIndex
     .BackColor.SchemeColor = Range("F2").Interior.ColorIndex
     .TwoColorGradient msoGradientHorizontal, 1
   End With
   ActiveChart.SeriesCollection(2).Select
   With Selection.Fill
     .ForeColor.SchemeColor = Range("A3").Interior.ColorIndex
     .BackColor.SchemeColor = Range("F3").Interior.ColorIndex
     .TwoColorGradient msoGradientFromCenter, 1
   End With
   ActiveChart.SeriesCollection(3).Select
   With Selection.Fill
     .ForeColor.SchemeColor = Range("A4").Interior.ColorIndex
     .BackColor.SchemeColor = Range("F4").Interior.ColorIndex
     .TwoColorGradient msoGradientDiagonalDown, 1
   End With
End Sub
```

Изменение параметров диаграммы

Изменять можно различные параметры диаграммы. Продемонстрируем, как это делается, на следующем примере. На рабочем листе в диапазоне **A2:B22** собраны некоторые данные, на основе которых построен график (рис. 10.9).



	A	B	C	D
1	X	Y		
2	-2,0	0,00000	Минимальное значение оси Y	-1
3	-1,8	1,90442	Максимальное значение оси Y	3
4	-1,6	2,43470	Цена основного деления оси Y	0,5
5	-1,4	1,86407	Цена промежуточного деления оси Y	0,1
6	-1,2	0,84641	Цвет графика	
7	-1,0	0,00000	Размер маркера	10
8	-0,8	-0,37618	Цвет фона маркера	
9	-0,6	-0,34238	Цвет маркера	
10	-0,4	-0,15217	Тощина графика	xlThick
11	-0,2	-0,02351	Тип маркера	xlMarkerStyleAutomatic
12	0,0	0,00000		
13	0,2	0,02351		
14	0,4	0,15217		
15	0,6	0,34238		
16	0,8	0,37618		
17	1,0	0,00000		
18	1,2	-0,84641		
19	1,4	-1,86407		
20	1,6	-2,43470		
21	1,8	-1,90442		
22	2,0	0,00000		

Рис. 10.9. Изменение параметров диаграммы

Пользователь может задать следующие параметры диаграммы, введя соответствующие значения в ячейки:

- минимальное значение оси Y — ячейка **D2**;
- максимальное значение оси Y — ячейка **D3**;
- цена основного деления оси Y — ячейка **D4**;
- цена промежуточного деления оси Y — ячейка **D5**;
- размер маркера — ячейка **D7**;
- ширина графика — ячейка **D10**;
- тип маркера — ячейка **D11**.

Кроме того, задав цвет следующих ячеек и нажав кнопку **Установить цвет,** пользователь может установить цвет следующих элементов:

- ячейка **D6** — графика;

- ячейка **D8** — фона маркера;
- ячейка **D9** — цвета маркера.

Для реализации этого проекта на рабочем листе дополнительно к графику:

- создайте кнопку и установите при помощи окна **Properties** значения ее свойств `Name` и `Caption` равными `cmdColor` и `Установить цвет`;
- в диапазон **G3:G6** введите допустимые значения (`xlHairline`, `xlThin`, `xlMedium`, `xlThick`) свойства `Weight` объекта `Border`, определяющие толщину графика. При помощи команды **Данные | Проверка** ячейке **D10** установите тип данных **Список**, а источник данных — **G3:G6**;
- в диапазон **H3:H8** введите допустимые значения (`xlMarkerStyleAutomatic`, `xlMarkerStyleCircle`, `xlMarkerStyleDash`, `xlMarkerStyleDiamond`, `xlMarkerStyleDot`, `xlMarkerStyleNone`) свойства `MarkerStyle` объекта `Series`, определяющие тип маркера. При помощи команды **Данные | Проверка** ячейке **D11** установите тип данных **Список**, а источник данных — **H3:H8**;
- в модуле рабочего листа наберите следующий код (листинг 10.11).

Листинг 10.11. Изменение параметров диаграммы

```
Private Sub cmdColor_Click()
   ActiveSheet.ChartObjects(1).Activate
   ActiveChart.SeriesCollection(1).Select
   With Selection
      .Border.ColorIndex = Range("D6").Interior.ColorIndex
      .MarkerBackgroundColorIndex = Range("D8").Interior.ColorIndex
      .MarkerForegroundColorIndex = Range("D9").Interior.ColorIndex
   End With
End Sub

Private Sub Worksheet_Change(ByVal Target As Range)
   If Not (Application.Intersect(Target, Range("D2:D11")) Is Nothing)
Then
      SetChart
   End If
End Sub

Private Sub SetChart()
   If ActiveSheet.ChartObjects.Count <> 1 Then Exit Sub
   ActiveSheet.ChartObjects(1).Activate
   ActiveChart.Axes(xlValue).Select
    With ActiveChart.Axes(xlValue)
```

```
        .MinimumScale = Range("D2").Value
        .MaximumScale = Range("D3").Value
        .MajorUnit = Range("D4").Value
        .MinorUnit = Range("D5").Value
    End With
    ActiveChart.SeriesCollection(1).Select
    With Selection
       .Border.Weight = SetWidthChart
       .MarkerSize = Range("D7").Value
       .MarkerStyle = SetMarkerStyle
   End With
End Sub

Function SetWidthChart() As Long
   Select Case Range("D10").Value
      Case "xlHairline"
         SetWidthChart = xlHairline
      Case "xlThin"
         SetWidthChart = xlThin
      Case "xlMedium"
         SetWidthChart = xlMedium
      Case "xlThick"
         SetWidthChart = xlThick
   End Select
End Function

Function SetMarkerStyle() As Long
   Select Case Range("D11").Value
      Case "xlMarkerStyleAutomatic"
         SetMarkerStyle = xlMarkerStyleAutomatic
      Case "xlMarkerStyleCircle"
         SetMarkerStyle = xlMarkerStyleCircle
      Case "xlMarkerStyleDash"
         SetMarkerStyle = xlMarkerStyleDash
      Case "xlMarkerStyleDiamond"
         SetMarkerStyle = xlMarkerStyleDiamond
      Case "xlMarkerStyleDot"
         SetMarkerStyle = xlMarkerStyleDot
   End Select
End Function
```

Печать и предварительный просмотр печати диаграммы

Печать диаграммы реализуется методом `PrintOut` объекта `Chart`, а предварительный просмотр — методом `PrintPreview`. В следующем примере демонстрируется применение этих методов (листинг 10.12). На рабочем листе расположена таблица и внедренная диаграмма. Кроме того, имеются две кнопки — **Печать диаграммы** и **Предварительный просмотр**. При нажатии первой кнопки производится печать диаграммы, а второй — предварительный просмотр (рис. 10.10).

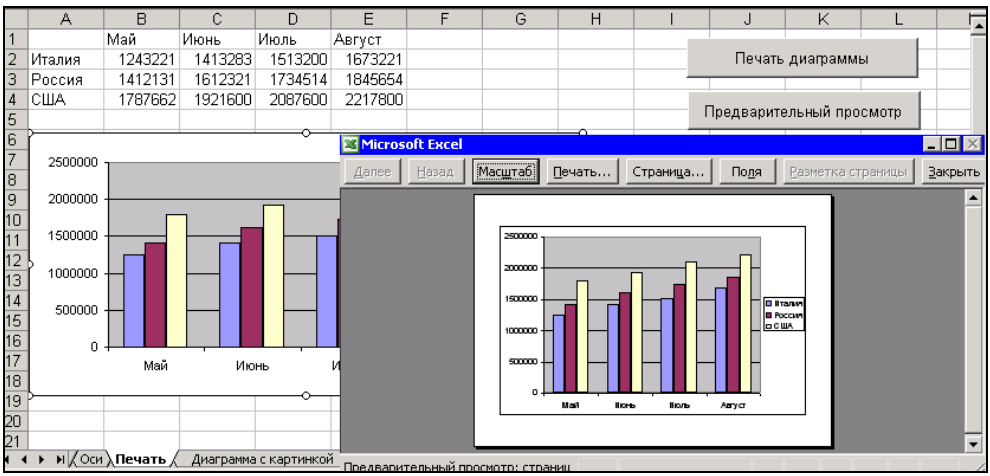


Рис. 10.10. Печать и предварительный просмотр печати диаграммы

Листинг 10.12. Печать и предварительный просмотр печати диаграммы

```
Private Sub cmdPrint_Click()
   ActiveSheet.ChartObjects(1).Chart.PrintOut
End Sub

Private Sub cmdPreview_Click()
   ActiveSheet.ChartObjects(1).Chart.PrintPreview
End Sub
```

Построение диаграммы на основе данных из нескольких листов

Диаграмму можно строить как на основе данных из одного рабочего листа, так и из нескольких. Следующий пример (листинг 10.13) демонстрирует, как это можно делать. Рабочая книга состоит из нескольких рабочих листов, например **Июнь**, **Июль**, **Август**, в диапазонах **A1:B4** содержатся данные о результатах продаж по странам (Италия, Россия и США) за указанные месяцы. При этом, как число рабочих листов, так и стран может быть произвольным, большим, равным или меньшим, чем три. На рабочем листе располагается кнопка **Построить диаграмму**, у которой значение свойства `Name` установлено равным `cmdChartBuilder`. Нажатие на эту кнопку приводит к созданию листа **Диаграмма** (с предварительным удалением предыдущей версии этого листа, если таковая существовала) и построению диаграммы на основе данных из всех рабочих листов (рис. 10.11).

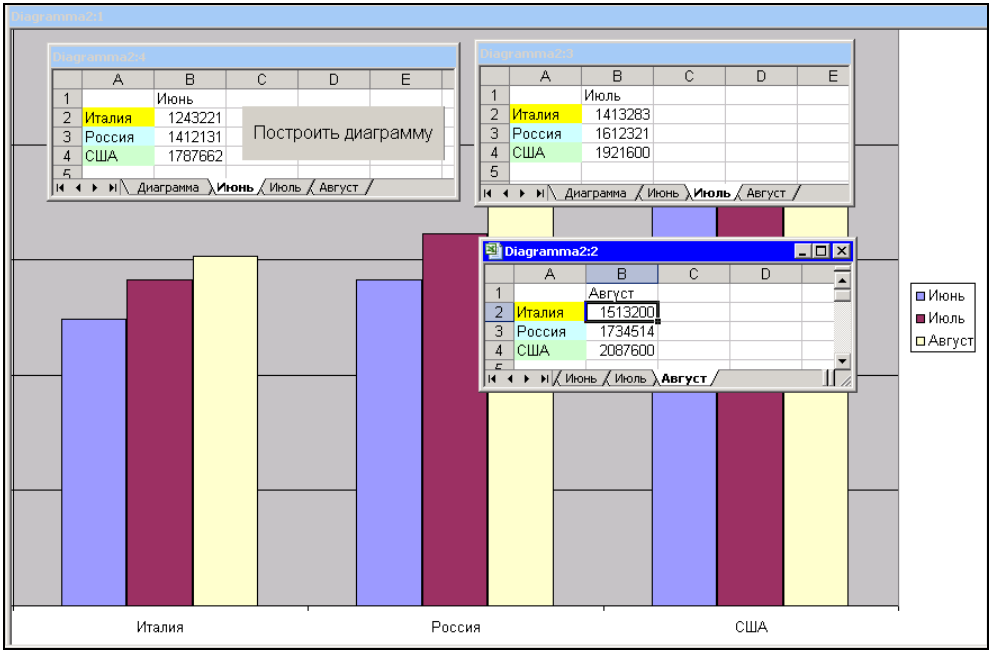


Рис. 10.11. Построение диаграммы на основе данных из нескольких листов

Для того чтобы при построении диаграммы не производилось мерцание экрана, перед ее построением при помощи свойства `ScreenUpdating` отключается обновление экрана, а по завершению конструирования — включается. Для отключения предупреждения об удалении листа диаграммы в коде используется свойство `DisplayAlerts`.

Листинг 10.13. Построение диаграммы на основе данных из нескольких листов

```
Const chartName As String = "Диаграмма"

Private Sub cmdChartBuilder_Click()
   Application.ScreenUpdating = False
   Application.DisplayAlerts = False
   ChartKiller
   ChartBuilder
   Application.ScreenUpdating = True
   Application.DisplayAlerts = True
End Sub

Private Sub ChartBuilder()
   Dim i As Integer, n As Integer
   Charts.Add
   ActiveChart.Location Where:=xlLocationAsNewSheet, Name:=chartName
   ActiveChart.ChartType = xlColumnClustered
   ActiveChart.HasLegend = True
   n = Range("A1").CurrentRegion.Rows.Count
   With Worksheets(1)
      ActiveChart.SetSourceData _
         Source:=.Range(.Cells(2, 1), .Cells(n, 2)), _
         PlotBy:=xlColumns
   End With
   For i = 1 To Worksheets.Count - 1
      ActiveChart.SeriesCollection.NewSeries
   Next
   For i = 1 To Worksheets.Count
      With Worksheets(i)
         ActiveChart.SeriesCollection(i).Values = _
            .Range(.Cells(2, 2), .Cells(n, 2))
         ActiveChart.SeriesCollection(i).Name = .Cells(1, 2)
      End With
   Next
End Sub

Private Sub ChartKiller()
   Dim ch As Chart
```

```
   For Each ch In Charts
      If ch.Name = chartName Then
         ch.Delete
         Exit Sub
      End If
   Next
End Sub
```

Защита диаграммы

Рабочий лист так же, как и лист диаграммы, можно защитить командой **Сервис | Защита | Защитить лист**. Кроме того, допустимы другие типы защиты диаграммы:

- `ProtectData` — устанавливает, возможно ли изменение диапазона, на основе которого конструируется диаграмма;
- `ProtectFormatting` — определяет, разрешено ли форматирование диаграммы;
- `ProtectSelection` — определяет, допустим ли выбор элементов диаграммы;
- `ProtectGoalSeek` — устанавливает, разрешено ли пользователю изменять данные путем перетаскивания элементов диаграммы.

Все эти типы защиты диаграммы можно устанавливать с помощью окна **Properties**.

Изменение диапазона, на основе которого конструируется диаграмма

В качестве примера установки защиты на диаграмму напишем приложение (рис. 10.12), в котором допустимо изменение диапазона, на основе которого конструируется диаграмма.

Итак, в проекте имеется рабочий лист **Защита**. В нем:

- в диапазон **A2:A13** введите список названий месяцев;
- в диапазон **B2:B13** введите размеры продаж некоторой фирмы за определенные месяцы;
- при помощи команды **Данные | Проверка** ячейке **F2** установите тип данных **Список**, а источник данных — **A2:A13**. В этой ячейке будет выбираться начальный месяц диаграммы доходов;
- при помощи команды **Данные | Проверка** ячейке **F3** установите тип данных **Список**, а источник данных — **A2:A13**. В этой ячейке будет выбираться конечный месяц диаграммы доходов;

- сконструируйте диаграмму на основе диапазона **A2:B13**;
- в модуле **ЭтаКнига** наберите код из листинга 10.14, *а*. Он установит защиту для пользователя на рабочий лист и графические объекты. Пользователь может только вводить данные в ячейки **B2:B13** и управлять работой листа с помощью кода;
- в модуле рабочего листа **Защита** наберите код из листинга 10.14, *б*. В нем реализована процедура обработки события `Change` объекта `Worksheet`. Также отслеживаются изменения в ячейках **F2** и **F3** и, при помощи метода `Find` объекта `Range`, определяется, каким ячейкам диапазона **A2:A13** соответствуют выбранные значения из списков ячеек **F2** и **F3.** Далее, на основе диапазона с границами в этих ячейках, перестраивается диаграмма.

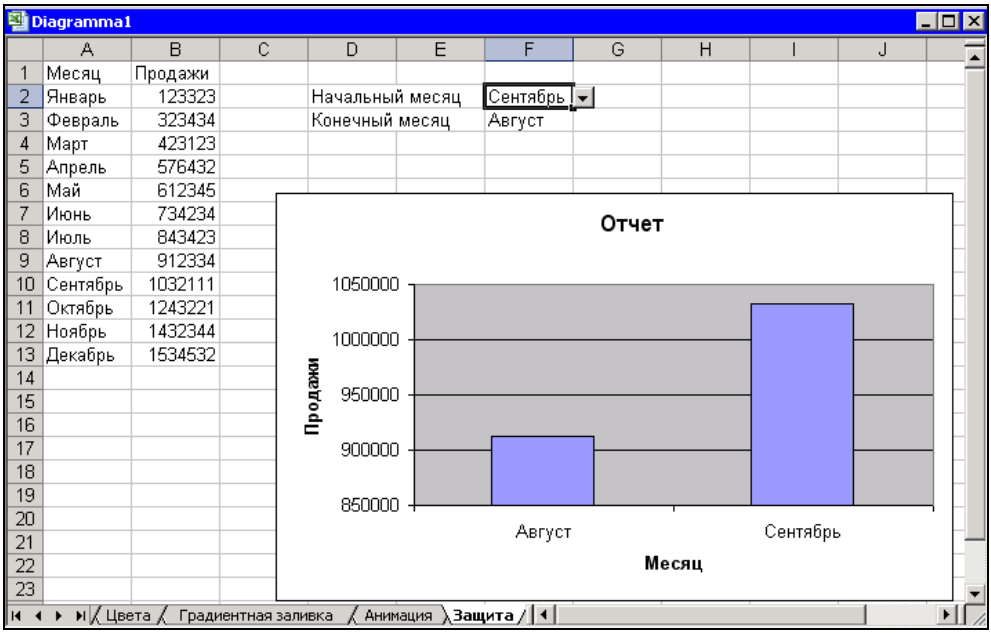


Рис. 10.12. Изменение диапазона, на основе которого конструируется диаграмма

Листинг 10.14, *а*. Защита диаграммы. Модуль ЭтаКнига

```
Private Sub Workbook_Open()
   SetProtection
End Sub

Private Sub SetProtection()
```

```
   Worksheets("Защита").Range("B2:B13").Locked = False
   Worksheets("Защита").Protect _
                        UserInterfaceOnly:=True
End Sub
```

Листинг 10.14, *б*.Защита диаграммы. Модуль рабочего листа

```
Private Sub Worksheet_Change(ByVal Target As Range)
   Dim adr As String
   adr = Target.Address(False, False, xlA1)
   If adr = "F2" Or adr = "F3" Then
      Dim c1 As Range, c2 As Range
      Set c1 = Range("A2:A13").Find( _
               What:=Range("F2").Value, LookIn:=xlValues)
      Set c2 = Range("A2:A13").Find( _
              What:=Range("F3").Value, LookIn:=xlValues)
      Dim c As Chart
      Set c = ActiveSheet.ChartObjects(1).Chart
      c.SetSourceData _
          Source:=ActiveSheet.Range(c1, c2.Offset(0, 1)), _
          PlotBy:=xlColumns
   End If
End Sub
```

Анимация диаграммы

Вывод диаграмм можно анимировать, выводя их последовательно кадр за кадром. Например, в следующем проекте на рабочем листе диапазон **A2:A22** введены значения аргумента x некоторой функции, зависящей от параметра a, а именно:

$$y = \mathrm{Sin}(\pi\ a\ x)\ \mathrm{Cos}(5\ \pi\ a\ x)a$$

Значения параметра вводятся в ячейку **D2** (рис. 10.13).

В ячейку **B2** введена формула, которая затем за маркер заполнения протащена на диапазон **B2:B22**, с тем, чтобы в этот диапазон вывести соответствующие значения функции.

```
=SIN(PI()*A2*$D$2)*COS(5*PI()*A2*$D$2)*$D$2
```

Кроме того, на рабочем листе расположена кнопка, у которой установлены значения свойств `Name` и `Caption` равными `cmdMovie` и `Анимация`. В модуле рабочего листа наберите код из листинга 10.15.

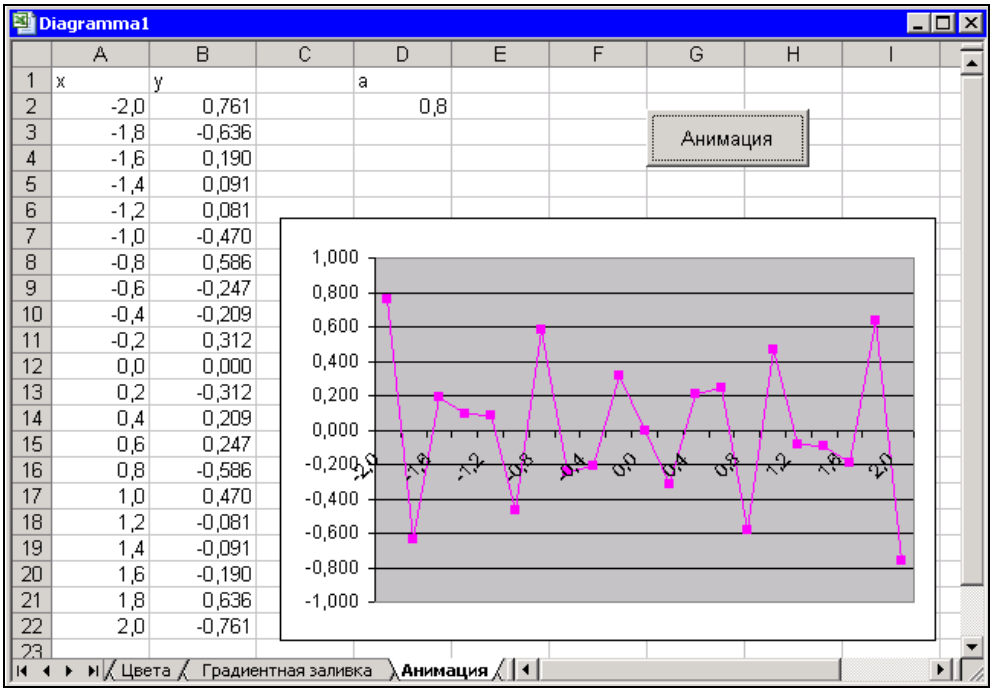


Рис. 10.13. Анимация диаграммы

Нажатие на кнопку приведет к последовательному выводу в ячейку **D2** различных значений параметра, что будет приводить к автоматическому обновлению диаграммы и создавать эффект анимации.

Листинг 10.15. Анимация диаграммы

```
Private Sub cmdMovie_Click()
   Static IsOn As Boolean
   IsOn = Not IsOn
   For x = -1 To 1 Step 0.2
     If Not IsOn Then Exit Sub
     Range("D2").Value = x
     Delay 1
  Next
End Sub

Private Sub Delay(TimeInterval As Single)
   Dim t As Single
```

```
   t = Timer
   Do
      DoEvents
   Loop While (Timer - t) < TimeInterval
End Sub
```

События и диаграммы

Диаграммы поддерживают обработку большого количества событий:

- `Activate` — активизация диаграммы;
- `BeforeDoubleClick` — двойной щелчок на диаграмме. Событие генерируется перед стандартной реакцией, т. е. выводом контекстного меню;
- `Calculate` — перевычисление данных диаграммы;
- `Deactivate` — деактивизация диаграммы;
- `DragOver` — перетаскивание над диаграммой диапазона;
- `DragPlot` — отпускание на диаграмму перетаскиваемого диапазона;
- `MouseDown` — нажатие кнопки мыши на диаграмме;
- `MouseMove` — перемещение мыши над диаграммой;
- `MouseUp` — отпускание кнопки мыши на диаграмме;
- `Resize` — изменение размеров диаграммы;
- `Select` — выделение элемента диаграммы;
- `SeriesChange` — изменение значений элемента диаграммы путем их перетаскивания.

Эти события по умолчанию ассоциируются с диаграммами, созданными на отдельных листах диаграмм. Например, следующий код из модуля диаграммы (листинг 10.16, *а*) обрабатывает событие `Select` и выводит информацию о выбранном элементе диаграммы.

Листинг 10.16, *а*. Информация об элементе диаграммы. Первый подход

```
Private Sub Chart_Select(ByVal ElementID As Long, _
            ByVal Arg1 As Long, ByVal Arg2 As Long)
   Select Case ElementID
      Case xlAxis
         MsgBox "Axis"
      Case xlAxisTitle
         MsgBox "AxisTitle"
```

```
      Case xlChartArea
         MsgBox "ChartArea"
      Case xlChartTitle
         MsgBox "ChartTitle"
      Case xlCorners
         MsgBox "Corners"
      Case xlDataLabel
         MsgBox "DataLabel"
      Case xlDataTable
         MsgBox "DataTable"
      Case xlFloor
         MsgBox "Floor"
      Case xlLegend
         MsgBox "xlLegend"
      Case xlMajorGridlines
         MsgBox "MajorGridlines"
      Case xlMinorGridlines
         MsgBox "MinorGridlines"
      Case xlPlotArea
         MsgBox "PlotArea"
      Case xlSeries
         MsgBox "Series"
      Case xlTrendline
         MsgBox "Trendline"
      Case xlWalls
         MsgBox "xlWalls"
   End Select
End Sub
```

Ту же задачу можно решить проще, не проводя идентификацию выбранного объекта с помощью параметра `ElementID`, а непосредственно возвращая имя выбранного объекта функцией `TypeName` (листинг 10.16, *б*).

Листинг 10.16, *б*. Информация об элементе диаграммы. Второй подход

```
Private Sub Chart_Select(ByVal ElementID As Long, _
            ByVal Arg1 As Long, ByVal Arg2 As Long)
  MsgBox TypeName(Selection)
End Sub
```

Привязка событий к вложенным в рабочий лист диаграммам

События явным образом не ассоциированы с вложенными в рабочий лист диаграммами. Для того чтобы их все-таки можно было обработать, надо проделать следующие подготовительные процедуры:

1. Создать модуль класса и назвать его, например, `MyEventClassModule`.
2. Объявить с ключевым словом `WithEvents` переменную типа `Chart` в модуле класса `MyEventClassModule` (листинг 10.17, *а*). После этого в редакторе кода модуля класса в списке объектов появится объект `MyEventClassModule`, а в списке событий — все связанные с диаграммами события.

Листинг 10.17, *а*. Привязка событий к вложенным в рабочий лист диаграммам. Модуль класса `MyEventClassModule`

```
Public WithEvents MyChartClass As Chart
```

3. В редакторе кода набрать код обработки тех событий, которых требует бизнес-логика вашего проекта. Например, в данном случае (листинг 10.17, *б*) при нажатии кнопки мыши отобразится предупреждающее сообщение **Не двигайте ее**.

Листинг 10.17, *б*. Привязка событий к вложенным в рабочий лист диаграммам. Модуль класса `MyEventClassModule`

```
Private Sub MyChartClass_MouseDown(ByVal Button As Long, _
      ByVal Shift As Long, ByVal x As Long, ByVal y As Long)
    MsgBox "Не двигайте ее"
End Sub
```

4. Остается только где-нибудь показать, что объект `MyChartClass` указывает на искомую вложенную диаграмму. Например, это можно сделать на этапе открытия книги, добавив в модуль **ЭтаКнига** следующий код (листинг 10.17, *в*), который ассоциирует первую из вложенных в первый рабочий лист диаграмм с объектом `MyChartClass`. Теперь при нажатии кнопки мыши на этой диаграмме отобразится сообщение **Не двигайте ее**.

Листинг 10.17, *в*. Привязка событий к вложенным в рабочий лист диаграммам. Модуль ЭтаКнига

```
Dim MyClassModule As New MyEventClassModule


Sub ChartInit()
```

```
   Set MyClassModule.MyChartClass = Worksheets(1).ChartObjects(1).Chart
End Sub

Private Sub Workbook_Open()
   ChartInit
End Sub
```

Изменение типа диаграммы при помощи контекстного меню

В качестве примера использования событий диаграмм приведем следующий проект, в котором при щелчке правой кнопкой мыши на диаграмме отображается контекстное меню, состоящее из трех команд: **Гистограмма**, **График** и **Круговая**, выбор которых и приводит к изменению типа диаграммы (рис. 10.14).

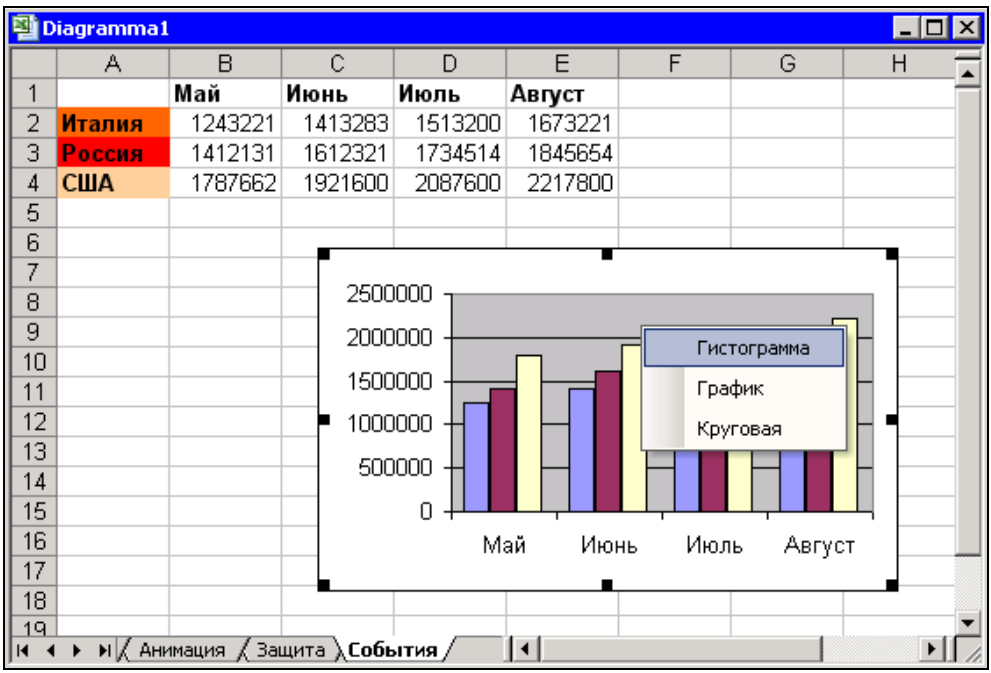


Рис. 10.14. Изменение типа диаграммы при помощи контекстного меню

Так как обрабатывается событие "щелчок правой кнопкой на внедренной диаграмме", то необходимо создать класс (в данном случае `ChartEventClass`), в котором и реализуется обрабатывающий код. При открытии рабочей книги надо связать экземпляр класса `ChartEventClass` с конкретной

диаграммой, создать контекстное меню, а также назначить командам этого меню соответствующие макросы, которые и осуществляют изменение типа диаграммы (листинги 10.18, *а*, *б*, *в*).

Листинг 10.18, *а*. Изменение типа диаграммы. Модуль ЭтаКнига

```
Dim ChartEvent As New ChartEventClass

Private Sub Workbook_Open()
   TbInit
   ChartInit
   PopUpBuilder
End Sub

Sub ChartInit()
   Set ChartEvent.ChartClass = _
      Worksheets("События").ChartObjects(1).Chart
End Sub

Sub TbInit()
   tb(0, 0) = "Гистограмма" : tb(0, 1) = xlColumnClustered
   tb(1, 0) = "График" : tb(1, 1) = xlLine
   tb(2, 0) = "Круговая" : tb(2, 1) = xlPie
End Sub

Private Sub PopUpBuilder()
   Dim btnGraph(2) As CommandBarButton
   Set cbPopUp = Application.CommandBars.Add(Name:="CustomBarPopup", _
                                             Position:=msoBarPopup)
   Dim i As Integer
   For i = 0 To 2
      With cbPopUp.Controls
         Set btnGraph(i) = .Add(Type:=msoControlButton)
         With btnGraph(i)
           .Caption = tb(i, 0)
           .Style = msoButtonCaption
           .OnAction = "Press" & i
         End With
      End With
   Next
```

```
End Sub

Private Sub Workbook_BeforeClose(Cancel As Boolean)
   Application.CommandBars("CustomBarPopup").Delete
End Sub
```

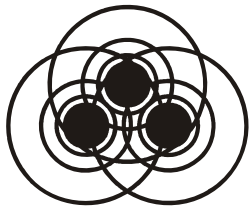
Листинг 10.18, *б*. Изменение типа диаграммы. Стандартный модуль

```
Public cbPopUp As CommandBar
Public tb(2, 1) As Variant

Sub Press0()
   ChangeTypeChart 0
End Sub

Sub Press1()
   ChangeTypeChart 1
End Sub

Sub Press2()
   ChangeTypeChart 2
End Sub

Sub ChangeTypeChart(idx As Integer)
   ActiveSheet.ChartObjects(1).Activate
   ActiveChart.ChartType = tb(idx, 1)
   If tb(idx, 1) = xlPie Then
      ActiveChart.HasLegend = True
   Else
      ActiveChart.HasLegend = False
   End If
End Sub
```

Листинг 10.18, *в*. Изменение типа диаграммы. Модуль класса `ChartEventClass`

```
Public WithEvents ChartClass As Chart

Private Sub ChartClass_BeforeRightClick(Cancel As Boolean)
   Cancel = True
   cbPopUp.ShowPopup
End Sub
```

Глава 11



Помощник MS Office

В MS Office имеется очень удобное средство — помощник, которое максимально приспособлено для отображения справочной информации и подсказок пользователю. Помощник отображается на экране командой **Справка | Показать помощника**, а скрывается с экрана командой **Справка | Скрыть помощника**. Параметры функционирования помощника задаются на вкладке **Параметры** окна **Помощник**, отображаемого при помощи выбора команды **Параметры** контекстного меню. Внешний вид помощника можно специфицировать при помощи вкладки **Коллекция** окна **Помощник**. В данной главе вы узнаете, как программно создавать помощника с надписями, с флажками, как проигрывать клипы помощника, как конструировать немодальное окно помощника и как добавлять в окно помощника значки.

Объект *Assistant*

Вся информация о помощнике инкапсулируется в объекте `Assistant`, который возвращается свойством `Assistant` объекта `Application`. Свойство `On` включает и отключает помощник, свойство `Visible` его визуализирует, а свойство `Animation` устанавливает проигрываемый клип. В следующем коде (листинг 11.1) отображается помощник, а затем последовательно проигрываются шестнадцать клипов. Итак, мотор включен, начали!

Листинг 11.1. Мультипликация помощника

```
Sub ShowMovie()
  Dim i As Integer
  For i = 0 To 15
     SelectFrame i
  Next
End Sub

Sub SelectFrame(num As Integer)
```

```
Dim a As Assistant
Set a = Application.Assistant
a.Top = Application.ActiveWindow.Top _
       + Application.ActiveWindow.Height / 2
a.Left = Application.ActiveWindow.Left _
        + Application.ActiveWindow.Width / 2
a.On = True
a.Visible = True
Select Case num
   Case 0
     a.Animation = msoAnimationAppear
   Case 1
     a.Animation = msoAnimationCheckingSomething
   Case 2
     a.Animation = msoAnimationBeginSpeaking
   Case 3
     a.Animation = msoAnimationCharacterSuccessMajor
   Case 4
     a.Animation = msoAnimationEmptyTrash
   Case 5
     a.Animation = msoAnimationGestureDown
   Case 5
     a.Animation = msoAnimationGestureLeft
   Case 6
     a.Animation = msoAnimationGestureRight
   Case 7
     a.Animation = msoAnimationGestureUp
   Case 8
     a.Animation = msoAnimationGetArtsy
   Case 9
     a.Animation = msoAnimationGetAttentionMajor
   Case 10
     a.Animation = msoAnimationGetAttentionMinor
   Case 11
     a.Animation = msoAnimationGetTechy
   Case 12
     a.Animation = msoAnimationGetWizardy
   Case 13
     a.Animation = msoAnimationGoodbye
```

```
      Case 14
        a.Animation = msoAnimationGreeting
      Case 15
        a.Animation = msoAnimationDisappear
   End Select
   Delay 3
End Sub

Private Sub Delay(TimeInterval As Single)
   Dim t As Single
   t = Timer
   Do
      DoEvents
   Loop While (Timer - t) < TimeInterval
End Sub
```

Объект *Balloon*

В объекте `Balloon` инкапсулированы структура помощника и его внешний вид. Новый объект `Balloon` создается свойством `NewBalloon` объекта `Assistant`. Метод `Show` отображает помощника. Свойство `BalloonType` задает тип объекта `Balloon`. По умолчанию значение этого свойства полагается равным `msoBalloonTypeButtons`. Допустимыми значениями свойства `BalloonType` являются следующие константы: `msoBalloonTypeBullets`, `msoBalloonTypeButtons` и `msoBalloonTypeNumbers`.

Свойство `Button` специфицирует те кнопки, которые отображаются в окне помощника и допустимыми значениями этого свойства являются следующие константы:

`msoButtonSetAbortRetryIgnore` `msoButtonSetBackClose`
`msoButtonSetBackNextClose` `msoButtonSetBackNextSnooze`
`msoButtonSetCancel` `msoButtonSetNextClose`
`msoButtonSetNone` `msoButtonSetOK`
`msoButtonSetOkCancel` `msoButtonSetRetryCancel`
`msoButtonSetSearchClose` `msoButtonSetTipsOptionsClose`
`msoButtonSetYesAllNoCancel` `msoButtonSetYesNo`
`msoButtonSetYesNoCancel`

Свойство `Icon` — выводимый значок. Допустимыми значениями свойства `Icon` являются следующие константы:

`msoIconAlert` `msoIconAlertCritical` `msoIconAlertInfo`
`msoIconAlertQuery` `msoIconAlertWarning` `msoIconNone`
`msoIconTip`

Свойство `Text` специфицирует текст, отображаемый в помощнике, а свойство `Heading` — его заголовок. У тех элементов помощника, которые имеют свойство `Text` (текстовая строка самого помощника, надписи и флажки) допустимо как раскрашивать фрагменты текста в 16 системных цветов, так и подчеркивать их. Для включения режима подчеркивания текста используются атрибуты `{ul}` или `{ul 1}`, а для отключения режима подчеркивания — атрибут `{ul 0}`. Для окрашивания фрагмента текста применяется атрибут `{cf number}`, допустимые значения параметра `number`, задающего цвет, перечислены в табл. 11.1.

Таблица 11.1. *Допустимые значения параметра* `number`*, задающего цвет*

Постоянная цвета	Цвет
0	Черный
1	Темно-красный
2	Темно-зеленый
3	Темно-желтый
4	Темно-голубой
5	Темно-розовый
6	Темно-синий
7	Темно-серый
248	Серый
249	Красный
250	Зеленый
251	Желтый
252	Голубой
253	Розовый
254	Синий
255	Белый

В следующем примере (листинг 11.2) создается помощник со специфицированным текстом, заголовком, значком и единственной кнопкой **OK**. Текст, отображаемый в помощнике, раскрашен в красный и темно-синий цвет, причем первая строка этого текста подчеркнута (рис. 11.1).

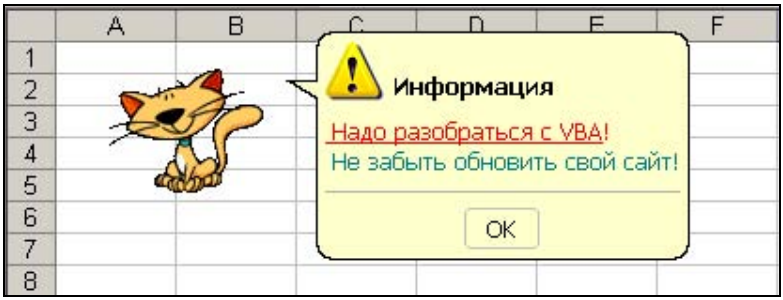


Рис. 11.1. Помощник

Листинг 11.2. Помощник

```
Sub CreareInfoBalloon()
   Dim b As Balloon
   Dim title As String
   Dim msg As String
   title = "Информация"
   msg = "{cf 249}{ul 1} Надо разобраться с VBA{ul 0}!" _
  & vbCr & "{cf 6} Не забыть обновить свой сайт!"
   With Application.Assistant
      .On = True
      .Visible = True
      Set b = .NewBalloon
   End With

   With b
      .BalloonType = msoBalloonTypeButtons
      .Button = msoButtonSetOK
      .Icon = msoIconAlert
      .Heading = title
      .Text = msg
      .Show
   End With
End Sub
```

Помощник с надписями

В объекте `Balloon`, как в родительском объекте, могут размещаться надписи (объекты `BalloonLabel`), которые образуют семейство `BalloonLabels`. Метод `Show` в этом случае возвращает индекс выбранной надписи, причем нумерация надписей производится с единицы. В следующем коде (листинг 11.3) создается помощник, который предлагает установить один из трех цветов ячеек (красный, желтый, зеленый) рабочего листа (рис. 11.2).

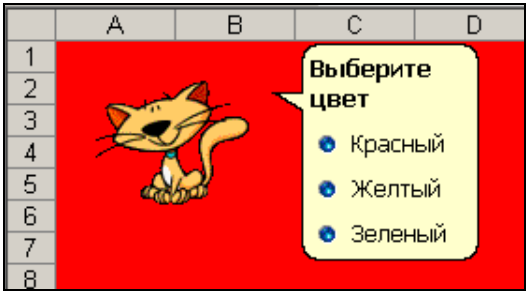


Рис. 11.2. Помощник с надписями

Листинг 11.3. Помощник с надписями

```
Sub SelectColor()
   Dim a As Assistant
   Set a = Assistant
   Dim b As Balloon
   With a
      .On = True
      .Visible = True
   Set b = .NewBalloon
   End With
   Dim choice As Integer
   With b
      .BalloonType = msoBalloonTypeButtons
      .Heading = "Выберите цвет"
      .Labels(1).Text = "Красный"
      .Labels(2).Text = "Желтый"
      .Labels(3).Text = "Зеленый"
      .Button = msoButtonSetNone
    choice = .Show
```

```
   End With
   MsgBox "Выбрали: " & b.Labels(choice).Text
   Select Case choice
      Case 1
         ActiveSheet.Cells.Interior.Color = RGB(255, 0, 0)
      Case 2
         ActiveSheet.Cells.Interior.Color = RGB(255, 255, 0)
      Case 3
         ActiveSheet.Cells.Interior.Color = RGB(0, 255, 0)
    End Select
    Set b = Nothing
    a.Visible = False
End Sub
```

Помощник с флажками

В объекте `Balloon`, как в родительском объекте, могут размещаться флажки (объекты `BalloonCheckBox`), которые образуют семейство `BalloonCheckBoxes`. Основным свойством объекта `BalloonCheckBox` является свойство `Checked`, которое определяет, установлен ли флажок. В следующем примере (листинг 11.4) создается помощник, в окне которого отображаются три флажка и две кнопки: **OK** и **Отмена** (рис. 11.3). Нажатие кнопки **OK** вызывает отображения диалогового окна с перечислением установленных флажков. То, какая кнопка нажата(**OK** и **Отмена**), идентифицируется по значению, возвращаемому методом `Show`, который и отображает окно помощника.

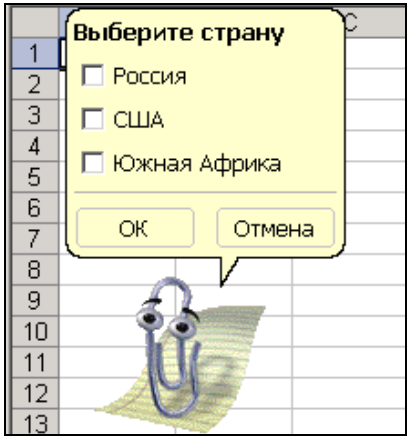


Рис. 11.3. Помощник с флажками

Листинг 11.4. Помощник с флажками

```
Sub SelectCountry()
   Dim a As Assistant
   Set a = Assistant
   Dim b As Balloon
   With a
      .On = True
      .Visible = True
   Set b = .NewBalloon
   End With
   With b
      .BalloonType = msoBalloonTypeButtons
      .Heading = "Выберите страну"
      .CheckBoxes(1).Text = "Россия"
      .CheckBoxes(2).Text = "США"
      .CheckBoxes(3).Text = "Южная Африка"
      .Button = msoButtonSetOkCancel
   End With

   If b.Show = msoBalloonButtonOK Then
      Dim i As Integer
      Dim msg As String
      msg = ""
      For i = 1 To 3
         If b.CheckBoxes(i).Checked Then
            msg = msg & b.CheckBoxes(i).Text & vbCr
         End If
      Next
      If Len(msg) = 0 Then
         MsgBox "No choice."
      Else
         MsgBox msg
      End If
   End If
   Set b = Nothing
   a.Visible = False
End Sub
```

Немодальное окно помощника

Окна помощника, рассмотренные в предыдущих разделах, были модальными. Но возможно создавать и немодальные окна, т. е. позволяющие продолжать работу с приложением и при открытом окне помощника. Для создания немодальных окон надо установить значение свойства `Mode` объекта `Balloon` равным `msoModeModeless`. Кроме того, на таком окне должна размещаться по крайней мере одна кнопка, и с ним при помощи свойства `Callback` должна быть ассоциирована процедура обработки нажатия кнопок в окне помощника. Эта процедура имеет предопределенную сигнатуру:

`NameOfProcedure``(`*`bln`* `As Balloon, `*`lbtn`* `As Long, `*`lPriv`* `As Long),`

где параметр *`bln`* возвращает ссылку на окно помощника, параметр *`lbtn`* возвращает цифровой код нажатой кнопки, а параметр *`lPriv`* содержит значение свойства `Private` объекта `Balloon`, однозначно идентифицирующее окно помощника. Немодальное окно закрывается методом `Close` объекта `Balloon`. Следующий код (листинг 11.5) демонстрирует работу с немодальным окном помощника. В нем при помощи помощника производится окраска фона активного листа рабочей книги (рис. 11.4).

Рис. 11.4. Окраска фона активного листа

Листинг 11.5. Немодальное окно Помощник с надписями

```
Sub SelectColor()
   Dim b As Balloon
   Set b = Assistant.NewBalloon
   With b
      .Heading = "Выберите цвет еще раз"
```

```
      .Labels(1).Text = "Красный"
      .Labels(2).Text = "Желтый"
      .Labels(3).Text = "Зеленый"
      .Button = msoButtonSetNone
      .BalloonType = msoBalloonTypeButtons
      .Mode = msoModeModeless
      .Callback = "SetColor"
      .Private = 123
      .Show
End With
End Sub

Private Sub SetColor(bln As Balloon, lbtn As Long, lPriv As Long)
   If lPriv = 123 Then
      Assistant.Animation = msoAnimationGestureDown
      Select Case lbtn
         Case 1
            ActiveSheet.Cells.Interior.Color = vbRed
         Case 2
            ActiveSheet.Cells.Interior.Color = vbYellow
         Case 3
            ActiveSheet.Cells.Interior.Color = vbGreen
       End Select
       bln.Close
       End If
End Sub
```

Добавление значка в надписи и флажки помощника

Значки на основе растровых файлов можно добавлять в те элементы помощника, у которых имеется свойство `Text`: в текстовую строку самого помощника, в надписи и флажки (рис. 11.5). При этом для спецификации файла, из которого загружается растровое изображение, используется следующий синтаксис:

`{type location sizing_factor}`, где:

- *`type`* — задает тип файла. Допустимы значения `bmp` и `wmf`;
- *`location`* — определяет местоположение файла;
- *`sizing_factor`* — задает ширину для WMF-файлов.

Как это делается, продемонстрировано в следующем примере (листинг 11.6). В нем при щелчке правой кнопки мыши отображается окно с предложением

одной из двух альтернатив, выбор которой приводит к вводу соответствующего текста в назначенную ячейку.

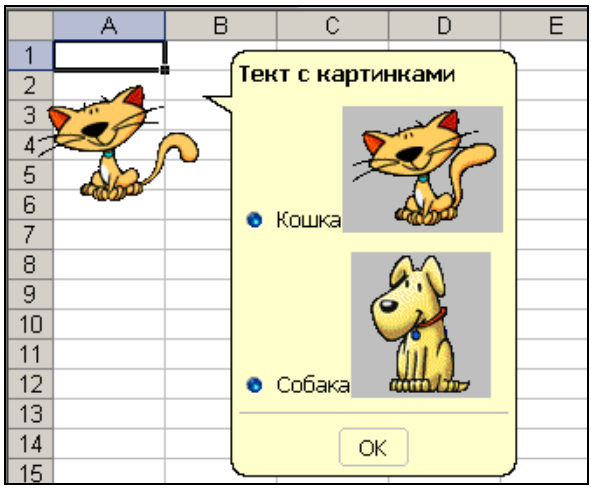
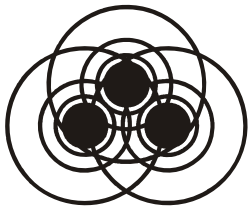


Рис. 11.5. Добавление значка в надписи и флажки помощника

Листинг 11.6. Добавление значка в надписи и флажки помощника. Модуль рабочего листа

```
Private Sub ShowAssistant()
   Dim bmpCat As String, bmpDog As String
   Dim b As Balloon
   Dim idx As Integer
   bmpCat = "{bmp c:\Cat.bmp}"
   bmpDog = "{bmp c:\Dog.bmp}"
   Set b = Assistant.NewBalloon
   With b
      .Heading = "Текст с картинками"
      .Labels(1).Text = "Кошка" & bmpCat
      .Labels(2).Text = "Собака" & bmpDog
      idx = .Show
   End With
   Select Case idx
      Case 1
         Selection.Value = "Кошка"
      Case 2
         Selection.Value = "Собака"
    End Select
End Sub
```

Глава 12



Работа с файлами

VBA позволяет обрабатывать файлы и работать с дисками и каталогами средствами объектной файловой системы (File System Object Model, FSO). FSO предоставляет разработчику возможность работы с файлами в ясной объектно-ориентированной манере. Основными объектами FSO являются:

`FileSystemObject`	`Drive`	`Folder`
`File`	`TextStream`	`Dictionary`

Кроме того, рассматривается поиск файлов с помощью объекта `FileSearch` и приводится список функций для работы с файлами.

Объект *FileSystemObject*

Объект `FileSystemObject` предоставляет доступ к файловой системе. Он обладает большим набором свойств и методов, управляющих параметрами файловой системы, а также доступ к трем семействам `Drives`, `Files` и `Folders`, предоставляющим доступ к дискам, файлам и каталогам файловой системы.

Создание объекта *FileSystemObject* с явной ссылкой на *Microsoft Scripting Runtime*

FSO опирается на библиотеку Microsoft Scripting Runtime Library (файл Scrrun.dll), которая совместно используется несколькими средами разработки и позволяет, в частности, расширить функции написания сценариев. Для использования этой библиотеки надо установить ссылку на библиотеку Microsoft Scripting Runtime в диалоговом окне **References**, которое отображается на экране выбором команды **Project | References**. После этого можно приступить к программированию объектов, функционирование которых обеспечивается этой библиотекой. Следующий код (листинг 12.1) демонстрирует конструирование подобным образом `FileSystemObject` объекта, на основе которого создается объект `TextStream`, используемый для построения текстового файла и ввода в него строки Hello, FSO!

Листинг 12.1. Создание текстового файла

```
Sub CreateFileSystemObject()
   Dim fs As FileSystemObject
   Set fs = New FileSystemObject
   Dim ts As TextStream
   Set ts = fs.CreateTextFile("c:\testfile.txt", True)
   ts.WriteLine ("Hello, FSO!")
   ts.Close
End Sub
```

Создание объекта *FileSystemObject* с неявной ссылкой на *Microsoft Scripting Runtime*

Объект `FileSystemObject` можно создать без явной ссылки на библиотеку Microsoft Scripting Runtime. Для этого надо воспользоваться функцией `CreateObject`, как продемонстрировано в следующем примере (листинг 12.2). Недостатком такого подхода является то, что переменные объявляются с использованием типа `Variant`, и поэтому в редакторе кода не будут отображаться сведения о списке свойств и методов соответствующих объектов.

Листинг 12.2. Создание текстового файла без явной ссылки на библиотеку Microsoft Scripting Runtime

```
Sub CreateFileSystemObject()
   Dim fs As Variant
   Set fs = CreateObject("Scripting.FileSystemObject")
   Dim ts As Variant
   Set ts = fs.CreateTextFile("c:\testfile.txt", True)
   ts.WriteLine ("Hello, FSO!")
   ts.Close
End Sub
```

Как проверить, существует ли диск

Для того чтобы проверить, существует ли диск, надо воспользоваться методом `DriveExists` объекта `FileSystemObject`, как сделано в листинге 12.3. Этот метод возвращает значение `True`, если специфицированный диск существует, и значение `False`, если не существует.

Листинг 12.3. Существует ли диск?

```
Sub CheckDrive()
   Dim fs As New FileSystemObject
   Dim dn As String
   dn = "c:"
   If fs.DriveExists(dn) Then
      MsgBox dn & " существует"
   Else
     MsgBox dn & " не существует"
   End If
End Sub
```

Получение информации о диске

Свойства объекта `Drive` позволяют собрать разнообразную информацию о диске, начиная от его размера и заканчивая используемой файловой системой. Следующий код (листинг 12.4) демонстрирует подобное применение объекта `Drive`. В результате использования этой программы на компьютере автора этой книги отобразилось сообщение, показанное на рис. 12.1.

Рис. 12.1. Получение информации о диске

Листинг 12.4. Получение информации о диске

```
Sub DriveInfo()
   Dim fs As New FileSystemObject
   Dim d As Drive
   Dim drvPath As String
   drvPath = "c:\"
   If fs.DriveExists(drvPath) Then
```

```
      Dim s As String
      Dim t As String
      Set d = fs.GetDrive(fs.GetDriveName(drvPath))
      s = "Drive " & UCase(drvPath) & " - "
      s = s & d.VolumeName & vbCr
      s = s & "Free Space: " & FormatNumber(d.FreeSpace / 1024, 0) _
        & " Kbytes" & vbCr
      s = s _
        & "Avaiable Space: " & FormatNumber(d.AvailableSpace / 1024, 0) _
        & " Kbytes" & vbCr
      s = s & "Avaiable Space: " & FormatNumber(d.TotalSize / 1024, 0) _
        & " Kbytes" & vbCr
      s = s & "File System: " & d.FileSystem & vbCr
      Select Case d.DriveType
         Case 0: t = "Unknown"
         Case 1: t = "Removable"
         Case 2: t = "Fixed"
         Case 3: t = "Network"
         Case 4: t = "CD-ROM"
         Case 5: t = "RAM Disk"
      End Select
      s = s & "Drive Type: " & t
      MsgBox s
   End If
End Sub
```

Как проверить, существует ли каталог или файл

Для того чтобы проверить, существует ли каталог или файл, надо воспользоваться методом `FolderExists` или `FileExists` объекта `FileSystemObject`. Эти методы возвращают значение `True`, если специфицированный каталог или файл существует, и значение `False`, если не существует. Приводимый ниже код (листинг 12.5) демонстрирует применение этих методов.

Листинг 12.5. Существование каталога или файла

```
Sub CheckFolder(folderName As String)
   ' Проверка существования каталога
   Dim fs As New FileSystemObject
   If fs.FolderExists(folderName) Then
      MsgBox folderName & " существует"
```

```
   Else
      MsgBox folderName & " не существует"
   End If
End Sub

Sub CheckFile(fileName As String)
   ' Проверка существования файла
   Dim fs As New FileSystemObject
   If fs.FileExists(fileName) Then
      MsgBox fileName & " существует"
   Else
      MsgBox fileName & " не существует"
   End If
End Sub
```

Создание и удаление каталога

Каталог создается методом `FolderExists`, а удаляется методом `DeleteFolder` объекта `FileSystemObject`. Следующий код (листинг 12.6) демонстрирует применение этих методов. В нем сначала создается каталог `MyFolder`, причем, прежде чем конструировать каталог, проверяется его существование, а затем по желанию пользователя этот каталог удаляется.

Листинг 12.6. Создание и удаление каталога

```
Sub DemoCreateDeleteFolder()
   Dim fs As New FileSystemObject
   Dim fl As Folder
   Dim fn As String
   fn = "c:\MyFolder"
   If Not fs.FolderExists(fn) Then
      Set fl = fs.CreateFolder(fn)
   End If
   Select Case MsgBox("Delete catalog " & fn, vbQuestion + vbYesNo)
      Case vbYes
          fs.DeleteFolder fn, False
          MsgBox "Файл удален"
      Case vbNo
         MsgBox " Файл не был удален"
   End Select
End Sub
```

Получение информации о каталоге

Свойства объекта `Folder` позволяют собрать разнообразную информацию о каталоге, его атрибутах. Следующий код (листинг 12.7) демонстрирует подобное применение объекта `Folder`. В результате использования этой программы на компьютере автора отобразилось сообщение, показанное на рис. 12.2.

Рис. 12.2. Получение информации о каталоге

Листинг 12.7. Получение информации о каталоге

```
Sub FolderInfo(folderName As String)
   Dim fs As New FileSystemObject
   If fs.FolderExists(folderName) Then
      Dim s As String
      Dim fl As Folder
      Set fl = fs.GetFolder(folderName)
      s = "Folder " & UCase(fnfolderName) & vbCr
      s = s & "Date Created: " & fl.DateCreated & vbCr
      s = s & "Date Last Accessed: " & fl.DateLastAccessed & vbCr
      s = s & "Date Last Modified: " & fl.DateLastModified & vbCr
      s = s & "Parent Folder: " & fl.ParentFolder.Name & vbCr
      s = s & "Path: " & fl.Path & vbCr
      s = s & "Size: " & FormatNumber(fl.Size / 1024, 0) _
              & " Kbytes" & vbCr
      If fl.Attributes And 0 Then s = s & "Normal attribute" & vbCr
      If fl.Attributes And 1 Then s = s & "Read only attribute" & vbCr
      If fl.Attributes And 2 Then s = s & "Hidden attribute" & vbCr
```

```
      If fl.Attributes And 4 Then s = s & "System attribute" & vbCr
      If fl.Attributes And 16 Then s = s & "Directory attribute" & vbCr
      If fl.Attributes And 32 Then s = s & "Archive attribute" & vbCr
      If fl.Attributes And 128 Then s = s & "Compressed attribute" & vbCr
      MsgBox s
   End If
End Sub

Sub Demo()
   FolderInfo "C:\windows"
End Sub
```

Копирование и перемещение файла

Копирование файла производится методом `CopyFile` объекта `FileSystemObject` либо методом `Copy` объекта `File`. Например, в следующем коде (листинг 12.8) производится копирование файла C:\Temp.txt в C:\TempNew.txt в случае, если последнего не существует.

Листинг 12.8. Копирование и перемещение файла

```
Sub DemoCopyFile()
   Dim fs As New FileSystemObject
   Dim fsrc As String
   Dim fdst As String
   fsrc = "c:\temp.txt"
   fdst = "c:\tempNew.txt"
   On Error GoTo errorhandler
   If fs.FileExists(fsrc) Then
      fs.CopyFile fsrc, fdst, False
   End If
   Exit Sub
errorhandler:
   MsgBox fdst & " exists"
End Sub
```

Удаление файла

Удаление файла производится методом `DeleteFile` объекта `FileSystemObject` и методом `Delete` объекта `File`. Следующий код (листинг 12.9) демонстрирует, как удаляется файл с предварительной проверкой его существования.

Листинг 12.9. Удаление файла

```
Sub DemoDeleteFile()
   Dim fs As New FileSystemObject
   f = "c:\temp.txt"
   If fs.FileExists(f) Then
      fs.DeleteFile f
   End If
End Sub
```

Информация о файле

Свойства объекта `File` позволяют собрать разнообразную информацию о файле, его местоположении и атрибутах. Следующий код (листинг 12.10) демонстрирует подобное применение объекта `File` для определения параметров рабочей книги с выполняемым кодом (рис. 12.3).

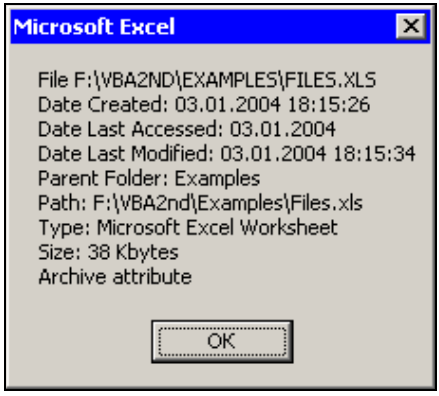


Рис. 12.3. Получение информации о рабочей книге

Листинг 12.10. Информация о файле

```
Sub FileInfo()
   Dim fs As New FileSystemObject
   Dim fn As String
   fn = ThisWorkbook.FullName
   If fs.FileExists(fn) Then
      Dim s As String
      Dim f  As File
      Set f = fs.GetFile(fn)
      s = "File " & UCase(fn) & vbCr
```

```
      s = s & "Date Created: " & f.DateCreated & vbCr
      s = s & "Date Last Accessed: " & f.DateLastAccessed & vbCr
      s = s & "Date Last Modified: " & f.DateLastModified & vbCr
      s = s & "Parent Folder: " & f.ParentFolder.Name & vbCr
      s = s & "Path: " & f.Path & vbCr
      s = s & "Type: " & f.Type & vbCr
      s = s & "Size: " & FormatNumber(f.Size / 1024, 0) _
              & " Kbytes" & vbCr
      If f.Attributes And 0 Then s = s & "Normal attribute" & vbCr
      If f.Attributes And 1 Then s = s & "Read only attribute" & vbCr
      If f.Attributes And 2 Then s = s & "Hidden attribute" & vbCr
      If f.Attributes And 4 Then s = s & "System attribute" & vbCr
      If f.Attributes And 16 Then s = s & "Directory attribute" & vbCr
      If f.Attributes And 32 Then s = s & "Archive attribute" & vbCr
      If f.Attributes And 128 Then s = s & "Compressed attribute" & vbCr
      MsgBox s
   End If
End Sub
```

Список всех файлов данного каталога

Свойство `Files` объекта `Folder` возвращает семейство `Files` всех файлов данного каталога. Поэтому, если требуется получить список файлов данного каталога, достаточно воспользоваться этим свойством, например, как это делается в листинге 12.11.

Листинг 12.11. Список всех файлов данного каталога

```
Sub ListFiles()
   Dim fs As New FileSystemObject
   Dim fl As Folder
   Dim fls As Files
   Dim f As File
   Dim s As String
   Set fl = fs.GetFolder("c:\")
   Set fls = fl.Files
   For Each f In fls
        s = s & f.Name & vbCr
   Next
   MsgBox s
End Sub
```

Открытие файла и создание *TextStream* объекта

Для открытия файла и создания `TextStream` объекта, представляющего собой текстовый поток ввода/вывода данных в текстовый файл, имеются метод `OpenAsTextStream` объекта `File` и два метода `CreateTextFile` и `OpenTextStream` объекта `FileSystemObject`, причем наиболее универсальным из этих методов является последний.

Метод *OpenAsTextStream*

Метод `OpenAsTextStream` объекта `File` открывает указанный файл и создает объект `TextStream`, обеспечивающий специфицированный доступ, а именно — для записи, добавления или считывания данных из файла. При попытке открыть несуществующий файл, метод генерирует ошибку.

`OpenAsTextStream([`*iomode*`, [`*format*`]])`, где:

- *iomode* — необязательный параметр, задающий то, для какой операции открывается файл. Допустимыми значениями являются следующие константы: `ForReading`, `ForWriting` и `ForAppending`;
- *format* — необязательный параметр, задающий тип файла. Допустимыми значениями являются следующие константы: `TristateUseDefault` (тип файла, используемый в системе по умолчанию), `TristateTrue` (Unicode-файл), `TristateFalse` (ASCII-файл).

В следующем коде (листинг 12.12) открывается существующий файл C:\Test.txt. Создается объект `TextStream` для записи в него строки "Hello, World!", после чего этот объект закрывается. Затем еще раз создается объект `TextStream`, теперь для добавления в него строки "Hello, World!", после чего этот объект закрывается. И, наконец, в третий раз создается объект `TextStream`, но теперь для считывания содержимого файла.

Листинг 12.12. Создание файла с помощью потока `TextStream`

```
Sub DemoOpenAsTextStream()
   Dim fs As New FileSystemObject
   Dim f As File
   Dim fn As String
   fn = "c:\test.txt"
   If fs.FileExists(fn) Then
      Set f = fs.GetFile(fn)
      Dim ts As TextStream
      Set ts = f.OpenAsTextStream(ForWriting) ' Запись данных
      ts.WriteLine "Hello, World!"
      ts.Close
```

```
      Set ts = f.OpenAsTextStream(ForAppending) ' Добавление данных
      ts.WriteLine "Hello, again!"
      ts.Close
      Set ts = f.OpenAsTextStream(ForReading) ' Считывание данных
      Dim msg As String
      msg = ts.ReadAll
      MsgBox msg
      ts.Close
   Else
      MsgBox "No file"
   End If
End Sub
```

Метод *CreateTextFile*

Метод `CreateTextFile` объекта `FileSystemObject` создает файл и объект `TextStream`, обеспечивающий доступ к данным файла. В этом методе не уточняется операция, для которой открыт файл.

`CreateTextFile(`*`filename`*`[, `*`overwrite`*`[, `*`unicode`*`]])`, где:

- *`filename`* — обязательный параметр, задающий имя создаваемого файла;
- *`overwrite`* — необязательный параметр, определяющий, надо ли перезаписывать файл, если он уже существует. Если значение равно `True`, то перезаписывается, если `False`, то не перезаписывается. При существовании файла на диске и значении параметра, равном `False`, метод генерирует ошибку;
- *`unicode`* — определяет код создаваемого текстового файла. Если значение параметра равно `True`, то файл создается как Unicode-файл. Если значение параметра равно `False`, то файл создается как ASCII-файл.

В следующем коде (листинг 12.13) создается файл C:\Testfile.txt для записи в него строки "Hello, World!". Так как при существовании файла на диске и значении параметра *`overwrite`*, равном `False`, метод `CreateTextFile` генерирует ошибку, то при использовании этого метода можно заранее не проверять существование файла, а ограничиться обработкой возможной ошибки.

Листинг 12.13. Создание файла методом `CreateTextFile`

```
Sub DemoCreateTextFile()
   Dim fs As New FileSystemObject
   Dim ts As TextStream
   On Error GoTo errorhandler
```

```
   Set ts = fs.CreateTextFile("c:\testfile.txt", False)
   ts.WriteLine "Hello, World!"
   ts.Close
   Exit Sub
errorhandler:
   MsgBox "File already exists"
End Sub
```

Метод *OpenTextStream*

Метод `OpenTextStream` объекта `FileSystemObject` открывает файл и создает объект `TextStream`, обеспечивающий доступ к файлу.

`OpenTextFile(`*`filename`*`[, `*`iomode`*`[, `*`create`*`[, `*`format`*`]]])`, где:

- *`filename`* — обязательный параметр, идентифицирующий имя открываемого файла;
- *`iomode`* — необязательный параметр, задающий, для какой операции открывается файл. Допустимыми значениями являются следующие константы: `ForReading`, `ForWriting` и `ForAppending`;
- *`create`* — необязательный параметр, который определяет, надо ли создавать новый файл, определяемый параметром *`filename`*, если он не существует;
- *`format`* — необязательный параметр, задающий тип файла. Допустимыми значениями являются следующие константы: `TristateUseDefault` (тип файла, используемый в системе по умолчанию), `TristateTrue` (Unicode-файл), `TristateFalse` (ASCII-файл).

Например, в следующем коде (листинг 12.14) открывается файл C:\Testfile.txt для записи данных и создается объект `TextStream` для доступа к его данным, причем если такового файла не существует, то он создается. В файл вводится всего одна строка "Hello, World!", после чего происходит закрытие объекта `TextStream`.

Листинг 12.14. Открытие файла для записи данных

```
ub DemoOpenTextFile()
   Dim fs As New FileSystemObject
   Dim ts As TextStream
   Set ts = fs.OpenTextFile("c:\testfile.txt", ForAppending, True)
   ts.WriteLine "Hello, World!"
   ts.Close
End Sub
```

Запись, присоединение и считывание данных из файла

В качестве примера записи, присоединения и считывания данных из файла приведем программу (листинг 12.15), которая создает таблицу табуляции функции Sin(*i*) при *i*, изменяющемся от 1 до 19. При этом сначала, для того чтобы ввести результат табуляции при изменении *i* от 1 до 9, создается, если его еще не было, файл C:\Table.txt. Затем файл повторно открывается, для того чтобы присоединить оставшиеся значения, и в заключение файл еще раз открывается, но уже для считывания данных. Для корректной работы программы не забудьте установить ссылку на **Microsoft Scripting Runtime** в диалоговом окне **References**, которое отображается на экране выбором команды **Project | References**.

Листинг 12.15. Запись, присоединение и считывание данных из файла

```
Sub DemoOperations()
   Dim fs As New FileSystemObject
   Dim ts As TextStream
   Dim i As Integer, j As Integer
   Set ts = fs.OpenTextFile("C:\Table.txt", ForWriting, True)
   For i = 1 To 9
      ts.Write (i)
      ts.Write (vbTab)
      ts.Write (FormatNumber(Sin(i), 3))
      ts.WriteLine
   Next
   ts.Close

   Set ts = fs.OpenTextFile("C:\Table.txt", ForAppending, False)
   For i = 10 To 19
      ts.WriteLine (i & vbTab & FormatNumber(Sin(i), 3))
   Next
   ts.Close

   Set ts = fs.OpenTextFile("C:\Table.txt", ForReading, False)
   Do While Not ts.AtEndOfStream
       Debug.Print ts.ReadLine
   Loop
   ts.Close
End Sub
```

Функции по работе с файлами

В VBA, помимо FSO, имеется ряд функций и операторов по работе с файлами и каталогами, дублирующих некоторые функции FSO. Одним из удобств этих функций является то, что для их применения не требуется устанавливать ссылку на используемую библиотеку. В табл. 12.1 эти функции и перечислены.

Таблица 12.1. *Функции и операторы по работе с файлами и каталогами*

Функции и операторы	Описание
`ChDir`	Изменяет текущую папку. `ChDir` *`Path`*
`ChDrive`	Изменяет текущий диск. `ChDrive` *`Drive`*
`CurDir`	Возвращает текущую папку
`Dir`	Возвращает файл или папку, соответствующую образцу, указанному в параметре *`PathName`*. Если несколько файлов соответствуют образцу, то возвращается первый из них. Используется для проверки существования файла на диске. `Dir[[(`*`PathName`*`[, `*`Attributes`*`])]`
	Здесь параметр *`Attributes`* может принимать следующие значения: `vbNormal` или `0` (обычный файл), `vbHidden` или `2` (скрытый), `vbSystem` или `4` (системный), `vbVolume` или `8` (метка тома, если это значение указано, то все остальные атрибуты игнорируются), `vbDirectory` или `16` (каталог).
	В коде допустим вызов процедуры `Dir` без параметров для повторного поиска файла
`FileAttr`	Режим работы файла
`GetAttr`	Возвращает значение типа `Integer`, определяющее атрибуты файла или каталога
`SetAttr`	Устанавливает атрибуты файла
`FileCopy`	Копирует файл
`FileDateTime`	Возвращает дату и время последнего изменения файла
`Kill`	Удаляет существующий файл. При этом удаляемый файл не помещается в корзину
`MkDir`	Создает новую папку
`RmDir`	Удаляет существующий каталог. При этом удаляемый каталог не помещается в корзину

Просмотр файлов в каталоге

Оператор `Dir` позволяет просмотреть все файлы в каталоге. Только использование его несколько специфично. Сначала необходимо вызвать `Dir` с параметрами и получить ссылку на первый файл, далее он вызывается уже без параметров для получения ссылки на очередной файл до тех пор, пока не вернется пустое имя файла, например, как это делается в листинге 12.16.

Листинг 12.16. Просмотр файлов в каталоге

```
Sub DemoDir()
   Dim s As String
   s = Dir("c:\windows\inf\*.*")
   Debug.Print s
      Do While s <> ""
         s = Dir
         Debug.Print s
   Loop
End Sub
```

Поиск файла и объект *FileSearch*

Объект `FileSearch` предоставляет функциональные возможности окна **Открытие документа** по поиску файла. С данным объектом тесно связаны следующие семейства объектов:

- `FileTypes` — семейство всех типов файлов, среди которых производится поиск;
- `FoundFiles` — семейство всех найденных файлов;
- `PropertyTests` — семейство всех критериев поиска;
- `SearchScopes` — семейство подкаталогов, в которых производится поиск.

У объекта `FileSearch` имеется обширная коллекция свойств, перечисленных в табл. 12.2, способных задать различные параметры поиска, и всего три метода, приведенные в табл. 12.3, которые предназначены для инициализации и запуска поиска.

***Таблица 12.2.** Свойства объекта `FileSearch`*

Свойство	Описание
`FileName`	Имя искомого файла, которое может включать символы "*" и "?"
`FileType`	Тип искомого файла

Таблица 12.2 *(окончание)*

Свойство	Описание
FileTypes	Возвращает список допустимых типов файла
FoundFiles	Возвращает объект FoundFiles, содержащий имена всех найденных файлов
LastModified	Возвращает константу, указывающую время последней модификации искомого файла. Допустимы значения: msoLastModifiedAnyTime, msoLastModifiedLastMonth, msoLastModifiedLastWeek, msoLastModifiedThisMonth, msoLastModifiedThisWeek, msoLastModifiedToday, msoLastModifiedYesterday
LookIn	Возвращает каталог, в котором производится поиск
MatchAllWordForms	Критерий сравнения слов при поиске файла, содержащем указанное слово
MatchTextExactly	Поиск файла, содержащего указанное слово
PropertyTests	Возвращает семейство PropertyTests, содержащее все критерии поиска файла
SearchFolders	Возвращает семейство SearchFolders, содержащее все каталоги, в которых производится поиск
SearchScopes	Возвращает семейство SearchScopes
SearchSubFolders	Устанавливает, надо ли производить поиск в подкаталогах
TextOrProperty	Задает слово или фразу, которая ищется в файле

Таблица 12.3. *Методы объекта* `FileSearch`

Метод	Описание
Execute	Начать поиск указанного файла
NewSearch	Установить у всех параметров поиска значения, подразумеваемые по умолчанию
RefreshScopes	Обновить текущий список достижимых объектов RefreshScope

Следующий код (листинг 12.17) демонстрирует, как можно произвести поиск файлов с указанным расширением в заданном каталоге и всех его подкаталогах.

Листинг 12.17. Поиск файла

```
Sub DemoFileSearch()
   With Application.FileSearch
      .NewSearch
```

```
      .LookIn = "C:\Windows"
      .SearchSubFolders = True
      .Filename = "*.bmp"
      .MatchTextExactly = True
      .FileType = msoFileTypeAllFiles
      If .Execute() > 0 Then
         Debug.Print "Найдено " & .FoundFiles.Count & " файлов"
         Dim i As Integer
         For i = 1 To .FoundFiles.Count
            Debug.Print .FoundFiles(i)
         Next
      Else
         Debug.Print "Файлы не найдены"
      End If
   End With
End Sub
```

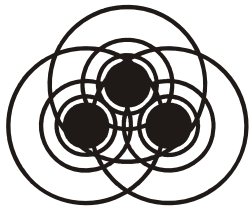
Как определить, существует ли рабочая книга

Объект `FileSearch` можно использовать для определения, существует ли данная рабочая книга. Для этого надо сравнить значение, возвращаемое методом `Execute`, с нулем, например, как это делается в листинге 12.18. Если оно положительно, то книга существует, а если ноль — то не существует.

Листинг 12.18. Определение, существует ли рабочая книга

```
Function WorkBookExists(FilePath As String, _
                        Filename As String) As Boolean
   With Application.FileSearch
      .LookIn = FilePath
      .Filename = Filename
      DoesWorkBookExist = .Execute > 0
   End With
End Function
```

Глава 13



Обработка ошибок и отладка программ

При составлении приложений важно предусмотреть, чтобы программа анализировала возможные ошибки, возникающие при ее выполнении по вине пользователя, и информировала его об этом, подсказывая пользователю, что конкретно он сделал неправильно. При этом возможно два подхода:

- *предотвращающий ошибки.* Программно анализируются вводимые или вычисляемые данные, и в случае, если они могут приводить к ошибке, обеспечивается, чтобы программа информировала пользователя о необходимости корректного задания данных;
- *обрабатывающий ошибки.* В случае появления ошибки, она перехватывается, обрабатывается и создается программный отклик на возникшую ошибку.

При создании приложений надо сочетать оба подхода, применяя в каждом конкретном случае и для каждой возможной ошибки тот подход, который кажется разработчику наиболее эффективным.

Разработка процедур, предотвращающих появление ошибок

Рассмотрим процесс создания приложения, в котором предотвращается появление ошибок, на простейшем примере. В диалоговом окне имеется только три поля: **a** — для ввода числителя, **b** — для ввода знаменателя и **c=a/b** — для отображения результата деления числителя на знаменатель по нажатию кнопки **OK** (рис. 13.1).

Для реализации приложения создайте форму с тремя надписями, тремя полями и кнопкой. Установите значения свойств элементов управления, как показано в табл. 13.1.

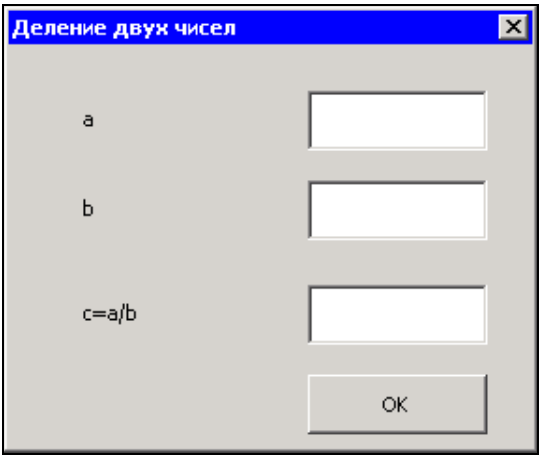


Рис. 13.1. Диалоговое окно **Деление двух чисел**

Таблица 13.1**. Значения свойств, установленные в окне **Properties

Элемент управления	Свойство	Значение
Надпись	Caption	a
Поле ввода	Name	txtA
Надпись	Caption	b
Поле ввода	Name	txtB
Надпись	Caption	c=a/b
Поле ввода	Name	txtC
Кнопка	Name	cmdOK
	Caption	OK

Следующая программа (листинг 13.1) находит обратное значение без контроля над появлением возможных ошибок.

Листинг 13.1. Деление двух чисел без предупреждения о возможных ошибках

```
Private Sub cmdOK_Click()
   Dim a As Double, b As Double, c As Double
   a = CDbl(txtA.Text)
   b = CDbl(txtB.Text)
   c = a / b
   txtC.Text = CStr(c)
End Sub
```

Несмотря на то, что рассматриваемая ситуация очень простая, уже она таит в себе множество подводных камней. Например, если пользователь по невнимательности забудет ввести в поле **a** число, при нажатии кнопки **OK** произойдет аварийное прерывание выполнения программы с мало понятным сообщением о несоответствии типов, отображаемым в диалоговом окне **Microsoft Visual Basic** (рис. 13.2).

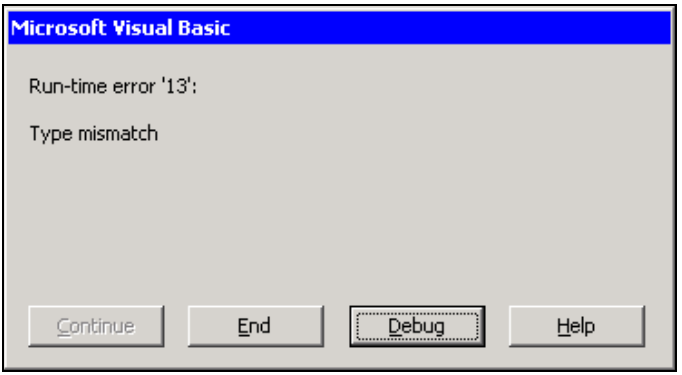


Рис. 13.2. Окно **Microsoft Visual Basic**

Данное сообщение об ошибке связано с инструкцией:

```
a = CDbl(txtA.text),
```

где параметром функции `CDbl` должна быть строка, преобразуемая в число. Если в поле **a** не введено ничего, по умолчанию из этого поля будет считываться пустая строка. Но пустая строка не может быть преобразована в число, и поэтому из-за функции `CDbl` происходит ошибка. Ошибка из-за несоответствия типов возникнет также, если в одно из полей пользователь по неосторожности введет число с десятичной запятой, а установками системы предусматривается десятичная точка и наоборот.

Данные ошибки ввода легко избежать, предусмотрев в программе предварительную проверку: преобразуются ли вводимые данные в числа. Эту предварительную проверку можно сделать, например, как показано в листинге 13.2. Кроме того, в этой процедуре заодно проверяется, не является ли значением знаменателя ноль. Деление на ноль тоже может вызвать ошибку. Программа информирует пользователя о возникшей ошибке, устанавливает фокус на поле с предполагаемым ошибочным значением и ожидает от пользователя корректного ввода числа.

Листинг 13.2. Деление двух чисел с проверкой корректности вводимых данных

```
Private Sub cmdOK_Click()
   Dim a As Double, b As Double, c As Double
   If Not IsNumeric(txtA.Text) Then
```

```
      MsgBox "Некорректный ввод значения a"
      txtA.SetFocus
      Exit Sub
   End If
   If Not IsNumeric(txtB.Text) Then
      MsgBox "Некорректный ввод значения b"
      txtB.SetFocus
      Exit Sub
   End If
   a = CDbl(txtA.Text)
   b = CDbl(txtB.Text)
   If b = 0 Then
      MsgBox "Введен ноль"
      txtB.SetFocus
      Exit Sub
   End If
   c = a / b
   txtC.Text = CStr(c)
End Sub
```

Контроль вводимых значений с помощью обработки события *KeyPress*

В примере из предыдущего раздела данные в поле ввода сначала вводились, а потом проводилась проверка корректности их ввода. Можно поступить иначе, проводя контроль непосредственно на этапе ввода и отсеивая неправильно вводимые значения. Этого можно достичь, добавив в программу код обработки события `KeyPress` поля ввода. Например, следующий код (листинг 13.3) позволяет вводить только цифры и нажимать клавиши <Delete> и <Back Space>. Действие всех других клавиш блокируется, потому что параметру `KeyAscii` процедуры обработки события `Keypress` устанавливается значение 0.

Листинг 13.3. Обеспечение ввода только целых положительных чисел

```
Private Sub txtA_Keypress(ByVal KeyAscii As MSForms.ReturnInteger)
   Select Case KeyAscii
      Case 8 ' Код клавиш <Delete> и <Back Space>
      Case 48 To 57 ' Код клавиш с <0> по <9>
      Case Else
         KeyAscii = 0
   End Select
End Sub
```

Перехват и обработка ошибок

Из диалогового окна **Microsoft Visual Basic** (см. рис. 13.2) видно, что каждая ошибка имеет свой код. В табл. 13.2 приведены коды наиболее часто встречаемых ошибок.

Таблица 13.2. *Коды наиболее часто встречаемых ошибок*

Код	Сообщение
5	Приложение не запущено
6	Переполнение
7	Не хватает памяти
9	Индекс выходит за пределы допустимого диапазона
11	Деление на ноль
13	Несоответствие типа
18	Произошло прерывание, вызванное пользователем
52	Неправильное имя файла или идентификатора
53	Файл не найден
54	Неверный режим работы с файлом
55	Файл уже открыт
56	Ошибка ввода/вывода
61	Переполнение диска
68	Устройство недоступно
71	Диск не готов
72	Повреждена поверхность диска
335	Невозможен доступ к системным ресурсам
368	Истек срок действия данного файла. Программе требуется файл более новой версии
482	Ошибка принтера

В идеале разрабатываемое приложение никогда не должно аварийно прерываться. В нем должны быть созданы средства перехвата любой возможной для этого приложения ошибки, обработки ее, выдачи сообщения пользователю и безаварийное продолжение работы. Обычно конструкция перехвата ошибок имеет следующую структуру:

Оператор `On Error`

Процедура обработки ошибки

Оператор `Resume`

Оператор *On Error*

Оператор `On Error` производит перехват ошибки. Он устанавливает, что программа должна делать в случае появления ошибки. Допустимы следующие варианты синтаксиса:

- `On Error GoTo` *`line`*

  Активизирует процедуру обработки ошибок, начало которой определяется обязательным параметром *`line`*. Его значением может быть либо метка строки, либо номер строки;

- `On Error Resume Next`

  Указывает, что при возникновении ошибки происходит передача управления на инструкцию, непосредственно следующую за инструкцией, где возникла ошибка;

- `On Error GoTo 0`

  Отключает любой активизированный обработчик ошибок в текущей процедуре, таким образом, инструкция `On Error Goto 0` отменяет действие инструкции `On Error Resume Next`.

Процедура обработки ошибки

Процедура обработки ошибки определяет тип возникшей ошибки и устанавливает, что программа должна делать в зависимости от типа ошибки.

Оператор *Resume*

Обеспечивает процедуре возможность продолжить работу после обработки ошибки. Допустимы три варианта синтаксиса этого оператора:

- `Resume`. После обработки ошибки управление передается той инструкции, в которой произошла ошибка;
- `Resume Next`. После обработки ошибки управление передается инструкции, следующей за инструкцией, в которой произошла ошибка;
- `Resume` *`line`*. После обработки ошибки управление передается инструкции, определенной параметром *`line`*. Значением этого параметра может быть любая метка строки или номер строки.

Объект *Err*

Процедура обработки ошибки обычно включает объект `Err`, который инкапсулирует в себе всю информацию о последней ошибке, возникшей во время выполнения приложения. В табл. 13.3 и 13.4 перечислены свойства и методы этого объекта.

Таблица 13.3. *Свойства объекта* `Err`

Свойство	Описание
`Number`	Возвращает код ошибки
`Source`	Имя программного файла, в котором возникла ошибка
`Description`	Возвращает строковое выражение, содержащее текст сообщения об ошибке, соответствующий коду ошибки
`HelpFile`	Полное имя (включая диск и путь) файла справки. Этот файл может быть вызван для поддержки сообщения об ошибке
`HelpContext`	Идентификатор раздела в справочном файле, указанном в свойстве `HelpFile`
`LastDLLError`	Содержит системный код ошибки для последнего вызова библиотеки динамической компоновки (DLL)

Таблица 13.4. *Методы объекта* `Err`

Метод	Описание
`Clear`	Очищает все значения свойств объекта `Err`. Метод `Clear` используется для явной очистки значений свойств объекта `Err` после завершения обработки ошибки. Это необходимо, например, при отложенной обработке ошибки, которая задается оператором `On Error Resume Next`
`Raise`	Создает ошибку выполнения. Используется при моделировании ситуаций ошибки. `Raise number, source, description, helpfile, helpcontext`, где: *`number`* — номер ошибки, т. е. целое число от 0 до 65535; *`source`* — строковое выражение, определяющее имя объекта или приложения, в котором возникла ошибка; *`description`* — строковое выражение, содержащее описание ошибки; *`helpfile`* — полное имя (включая диск и путь) файла справки Microsoft Windows, содержащего описание данной ошибки; *`Helpcontext`* — контекстный идентификатор, определяющий раздел в файле, указанном в параметре `Раздел`, соответствующий обрабатываемой ошибке

Принципиальным отличием применения объекта `Err` от подхода, использованного в предыдущем разделе, является то, что объект `Err` позволяет реагировать на возникшую в приложении ошибку, а не создавать средства, обеспечивающие проверку корректности данных до начала работы с ними. В этом смысле с помощью объекта `Err` возможно конструирование приложений, более гибко реагирующих на возникающие в них ошибки, причем

это происходит за счет сокращения и упрощения кода, что также является немаловажным достоинством объекта `Err`.

На конкретном примере покажем, как применяется объект `Err` при создании обработчика ошибок. В *разд. "Разработка процедур, предотвращающих появление ошибок"* в процессе создания диалогового окна **Деление двух чисел** и связанной с ним программы, на первый взгляд, были предусмотрены все возможные ошибки. Но это только на первый взгляд. Введем, например, в поле **b** величину `1E-310`. Это число ничем не лучше или не хуже любого другого числа типа `Double`. Тем не менее, вместо вывода результата произойдет аварийное прерывание выполнения программы с отображением сообщения об ошибке переполнения. Усовершенствуем нашу программу, предусмотрев возможность обработки как ранее описанных ошибок, так и ошибки переполнения (листинг 13.4, *а*).

Листинг 13.4, *а*. Обработчик ошибок

```
Private Sub cmdOK_Click()
   Dim a As Double, b As Double, c As Double
   On Error GoTo errorHandler1
   a = CDbl(txtA.Text)
   On Error GoTo errorHandler2
   b = CDbl(txtB.Text)
   c = a / b
   txtC.Text = CStr(c)
   Exit Sub
errorHandler1:
   MsgBox "Произошла ошибка: " & CStr(Err.Number) & "-" & _
                           Err.Description
   txtA.SetFocus
   Exit Sub
errorHandler2:
   Dim msg As String
   Select Case Err.Number
      Case 6
         msg = "Переполнение."
      Case 11
         msg = "На ноль делить нельзя"
      Case 13
         msg = "Должно быть число"
      Case Else
         msg = "Непредвиденная ошибка"
```

```
   End Select
   MsgBox "Произошла ошибка: " & CStr(Err.Number) & "-" & _
                              Err.Description & vbCr & msg
   txtB.SetFocus
   Exit Sub
End Sub
```

Приводимая выше программа имеет тонкую настройку на различные ошибки. При первоначальном этапе разработки программы можно было бы ограничиться более простым вариантом, в котором просто последовательно отлавливаются ошибки, информация о которых выводится средствами объекта `Err`, причем для вывода обработки очередной ошибки содержимое этого предварительно очищается методом `Clear`, например, как это делается в листинге 13.4, *б*.

Листинг 13.4, *б*. Обработчик ошибок. Второй вариант

```
Private Sub cmdOK_Click()
   Dim a As Double, b As Double, c As Double
   On Error GoTo errorHandler
   a = CDbl(txtA.Text)
   b = CDbl(txtB.Text)
   c = a / b
   txtC.Text = CStr(c)
   Exit Sub
errorHandler:
   MsgBox Err.Description & vbCr & Err.Number
   Err.Clear
   Resume Next
End Sub
```

В качестве примера использования оператора `Resume` при обработке ошибок приведем следующий код (листинг 13.5), в котором значения текущей ячейки данного диапазона увеличиваются на единицу. Если очередная ячейка содержит не число, то в этом случае генерируется ошибка, которая обрабатывается, и программа переходит к работе со следующей ячейкой.

Листинг 13.5. Последовательная обработка ячеек

```
Sub ResetValues()
   Dim c As Range
   On Error GoTo ErrorHandler
   For Each c In ActiveSheet.UsedRange
```

```
      'If c.Value <> 0 Then
         c.Value = c.Value + 1
      'End If
GoToNextValue:
   Next
   Exit Sub
ErrorHandler:
   If Err.Number = 13 Then ' Ошибка типа
      Err.Clear
      Resume GoToNextValue
   End If
End Sub
```

Создание пользовательских ошибок для тонкой настройки обработчика ошибок

Для тонкой настройки обработчика ошибок можно воспользоваться методом `Raise` объекта `Err`, который генерирует ошибку с кодом, заданным параметром `Number` и описанием — параметром `Description`. В следующем коде (листинг 13.6) мы опять возвращаемся к примеру с нахождением обратного значения, но теперь по проверке условия, является ли значение в поле числом, не происходит непосредственного отображения окна `Msgbox`, а производится генерация пользовательской ошибки и ее обработка, после которой отображается встроенное окно обработчика ошибки.

Листинг 13.6. Создание пользовательских ошибок

```
Private Sub cmdOK_Click()
   Dim a As Double, b As Double, c As Double
   On Error GoTo errorHandler
   If Not IsNumeric(txtA.Text) Then
      Err. Private Sub cmdOK_Click()
   Dim a As Double, b As Double, c As Double
   On Error GoTo errorHandler
   If Not IsNumeric(txtA.Text) Then
      Err.Raise Number:=60001, Description:="a не число"
   End If
   If Not IsNumeric(txtB.Text) Then
      Err.Raise Number:=60002, Description:="b не число"
```

```
   End If
   a = CDbl(txtA.Text)
   b = CDbl(txtB.Text)
   If b = 0 Then
      Err.Raise Number:=60003, Description:="b равно 0"
   End If
   c = a / b
   txtC.Text = CStr(c)
   Exit Sub
errorHandler:
   MsgBox Err.Description & " - ошибка " & Err.Number
   txtC.Text = ""
   Select Case Err.Number
      Case 60001
         txtA.SetFocus
      Case 60002
         txtB.SetFocus
      Case 60003
         txtB.SetFocus
      Case Else
         '
   End Select
End Sub Number:=60001, Description:="a не число"
   End If
   If Not IsNumeric(txtB.Text) Then
      Err.Raise Number:=60002, Description:="b не число"
   End If
   a = CDbl(txtA.Text)
   b = CDbl(txtB.Text)
   If b = 0 Then
      Err.Raise Number:=60003, Description:="b равно 0"
   End If
   c = a / b
   txtC.Text = CStr(c)
   Exit Sub
errorHandler:
   MsgBox Err.Description & " - ошибка " & Err.Number
   txtC.Text = ""
   Select Case Err.Number
```

```
      Case 60001
         txtA.SetFocus
      Case 60002
         txtB.SetFocus
      Case 60003
         txtB.SetFocus
      Case Else
         '
   End Select
End Sub
```

Отладка программ

При написании программ пользователь независимо от его опыта допускает те или иные ошибки. Кто-то из мудрых совершенно верно подметил, что только тот, кто ничего не делает, не совершает ошибок. Позвольте перефразировать эту жизненную мудрость для изучаемого в данной книге предмета следующим образом: в программах не делал ошибок только тот, кто никогда не писал кода. Итак, ошибки — это объективная неизбежность или реальное воплощение этой неизбежности. Какие же бывают ошибки, и как с ними бороться? Условно ошибки можно поделить на три типа: ошибки компиляции, выполнения и логические ошибки.

Ошибки компиляции

Ошибки компиляции возникают, если Visual Basic не может интерпретировать введенный код. Например, при некорректном вводе числа скобок, неправильном имени, неполном вводе инструкции и т. д. Некоторые из этих ошибок обнаруживаются Visual Basic при завершении набора строки с инструкцией в редакторе кода нажатием клавиши <Enter>. Строка, в которой содержится ошибка, выделяется красным цветом, и на экране отображается диалоговое окно с сообщением о возможной причине, вызвавшей ошибку (рис. 13.3).

Другие ошибки компиляции обнаруживаются перед выполнением программы. Отметим, что VBA каждый раз автоматически компилирует программу при ее запуске на выполнение. В этом случае предполагаемое местоположение ошибки выделяется синим цветом, и на экране отображается диалоговое окно **Microsoft Visual Basic** с сообщением о возможной причине, вызвавшей ошибку, например, опущена инструкция `End Select` в операторе `Select` (рис. 13.4).

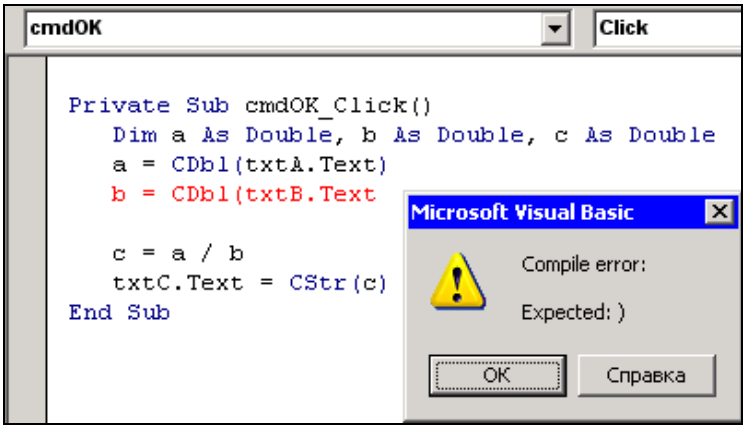


Рис. 13.3. Ошибка компиляции, обнаруженная при вводе инструкции

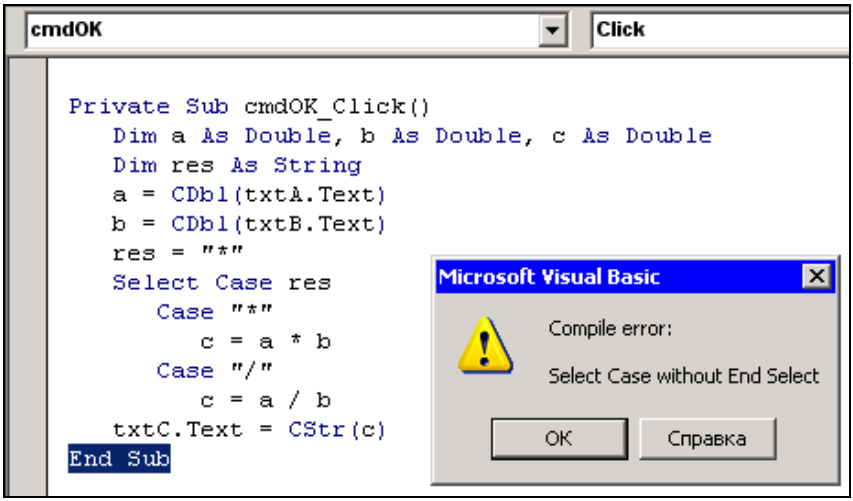


Рис. 13.4. Сообщение об ошибке компиляции в диалоговом окне **Microsoft Visual Basic**

Ошибки выполнения

Ошибки выполнения возникают после успешной компиляции программы при ее выполнении. Причинами таких ошибок могут быть, например:

- некорректная информация при считывании файла с диска;
- некорректные данные, введенные пользователем, например, требуется число, а пользователь вводит строковую информацию;
- некорректность вычислений, например, деление на ноль и т. д.

В этом случае на экране отображается диалоговое окно **Microsoft Visual Basic** с сообщением о номере ошибки и возможной причине, ее вызвавшей (рис. 13.5).

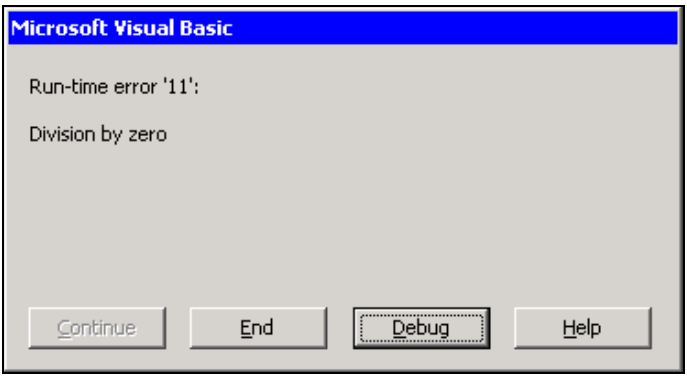


Рис. 13.5. Сообщение об ошибке выполнения в диалоговом окне **Microsoft Visual Basic**

Если в диалоговом окне **Microsoft Visual Basic** нажать кнопку **Debug**, то в строке модуля желтым цветом будет выделена строка, вызвавшая ошибку, и на выполнении которой выполнение программы было прервано. Кроме того, эта строка будет помечена стрелочкой. VBA перейдет в режим прерывания (рис. 13.6).

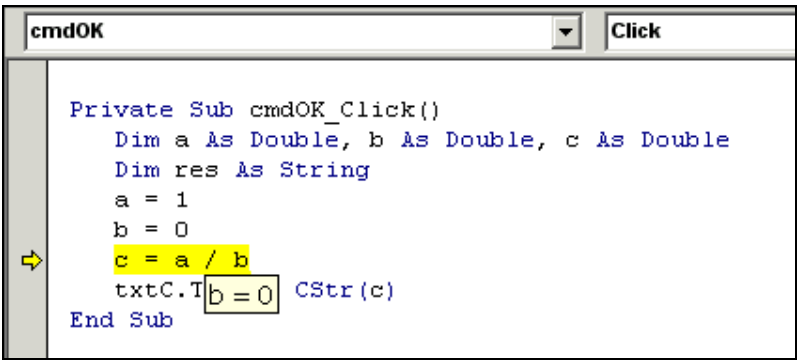


Рис. 13.6. Редактор кода в режиме прерывания

Одним из удобств режима прерывания является возможность узнать текущее значение переменных и свойств, для чего достаточно расположить указатель мыши на имени свойства или переменной. Это вызовет появление всплывающей подсказки с текущим значением переменной или свойства. В данном случае (см. рис. 13.6), видно, что значение переменной *b* равно 0, что

и вызвало ошибку. Для задания режима вывода всплывающей подсказки с текущими значениями данных должен быть установлен флажок **Auto Data Tips** вкладки **Editor** диалогового окна **Options**, вызываемого командой **Tools | Options**.

Кроме режима прерывания, приложение может находиться в режиме разработки и режиме выполнения. В режиме разработки, собственно, и создается приложение — конструируются формы, набирается код. В режиме выполнения приложение получает управление, и мы взаимодействуем с ним так же, как это будет делать и пользователь нашего приложения. Переключение между режимами работы удобно производить при помощи трех кнопок панели инструментов **Standard** (кстати, они также включены в панель инструментов **Debug**), перечисленных в табл. 13.5.

Таблица 13.5. *Кнопки панели инструментов* ***Standard****, управляющие режимом работы программы*

Кнопка	Название	Описание
	**Run Sub/UserForm**	Доступна в режиме конструирования. Переключает в режим выполнения. В режиме прерывания она также доступна, но играет роль кнопки **Continue**
	**Break**	Доступна в режиме выполнения. Переключает в режим прерывания
	**Reset**	Доступна в режиме выполнения. Переключает в режим разработки

Логические ошибки

Логические ошибки труднее всего обнаружить и устранить. Эти ошибки не приводят к прерыванию выполнения программы, т. е. визуально все идет гладко и выглядит так, как будто программа работает безупречно. Но это только кажущаяся идиллия, поскольку программа выдает неверные результаты. Локализация логических ошибок связана с тщательным анализом алгоритма программы с привлечением средств отладки VBA.

Директива *Option Explicit*

Простейшим средством предотвращения случайных ошибок является использование директивы `Option Explicit`. Эта директива предписывает объявлять все переменные, встречающиеся в программе. Использование директивы `Option Explicit` позволяет избежать следующей трудно отслеживаемой ошибки. Предположим, что в программе используется переменная с именем `Ссуда`, а при наборе имени этой переменной где-то в программе вместо

русской буквы "с" по ошибке набрана латинская буква "c". Визуально эти имена ничем не отличаются друг от друга, но воспринимаются программой как имена различных переменных. Если бы была использована инструкция `Option Explicit`, и переменная `Ссуда` была бы объявлена, то компилятор указал бы на переменную `Ссуда` с латинской буквой "c", как на необъявленную, и эта трудно отслеживаемая ошибка была бы быстро найдена.

Пошаговое выполнение программ

Редактор Visual Basic позволяет осуществлять пошаговое выполнение программы. Такой режим можно задать либо при помощи панели инструментов **Debug**, либо из меню **Debug**, которое включает команды и соответствующие комбинации клавиш (рис. 13.7). Если панель инструментов **Debug** не отображена на экране, то ее можно отобразить командой **View | ToolBars | Debug**.

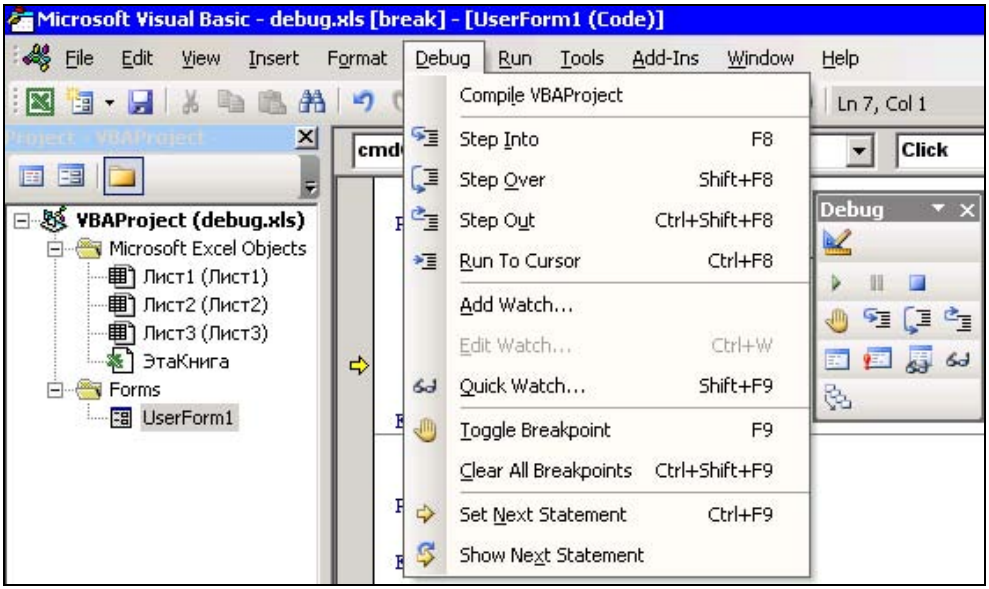


Рис. 13.7. Панель инструментов **Debug** и раскрывающееся меню **Debug**

Для выполнения программы в пошаговом режиме используются четыре команды:

- команда **Debug | Step Into** либо кнопка панели инструментов **Debug** осуществляют последовательную, шаг за шагом, отладку всей программы, включая процедуры, вызываемые в программе;
- команда **Debug | Step Over** либо кнопка панели инструментов **Debug** осуществляют последовательную, шаг за шагом, отладку всей программы,

не заходя в процедуры, вызываемые в программе. Если встречается процедура, то она выполняется целиком, а не шаг за шагом, как это делается в команде **Debug | Step Into**;

- команда **Debug | Step Out** либо кнопка  панели инструментов **Debug** завершают выполнение текущей процедуры и останавливаются на следующей инструкции программы, откуда процедура была вызвана;
- команда **Debug | Run to Cursor** выполняет программу до инструкции, помеченной курсором.

Точка прерывания

Visual Basic приостанавливает выполнение программы перед строкой кода, содержащей точку прерывания, и переключается в режим прерывания. Точка прерывания устанавливается или снимается командой **Debug | Toggle Breakpoint** либо кнопкой  панели инструментов **Debug.** В модуле точки прерывания выделяются полосой кирпичного цвета и кругом того же цвета (рис. 13.8).

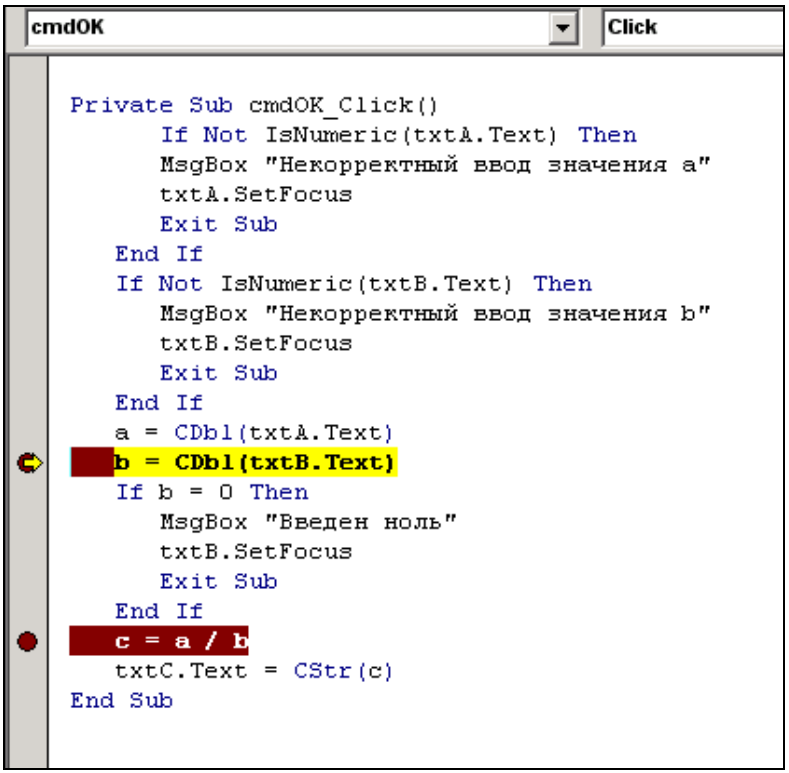


Рис. 13.8. Точки прерывания

В одном проекте может быть несколько точек прерывания. Все инструкции, расположенные выше, между и ниже точек прерывания, выполняются в обычном режиме.

Одновременно снять все точки прерывания можно командой **Debug | Clear All Breakpoint**.

Вывод значений свойств и переменных

Одним из удобств режима отладки является возможность узнать текущее значение переменных и свойств, для чего, как и в режиме прерывания, достаточно расположить указатель мыши на имени свойства или переменной. Это вызовет появление всплывающей подсказки с текущим значением переменной или свойства.

Окно *Watches*

Другим способом отслеживания текущих значений данных является использование диалогового окна **Watches**, которое отображается на экране либо командой **View | Watch Window**, либо командой **Debug | Quick Watch** (рис. 13.9). Диалоговое окно **Watches** позволяет одновременно отображать текущие значения нескольких переменных или свойств. Команда **Debug | Add Watch** добавляет новые контрольные значения в диалоговое окно **Watches**.

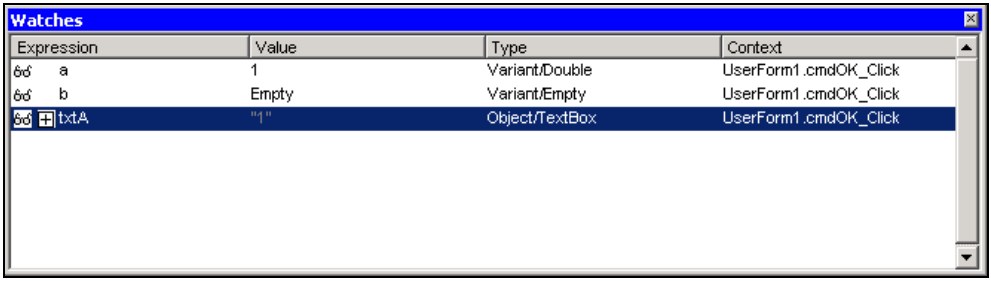


Рис. 13.9. Окно **Watches**

Удаление контрольного значения из диалогового окна производится его выделением и нажатием клавиши <Delete>.

Окно *Locals*

Окно **Locals**, отображаемое на экране командой **View | Locals Window**, выводит значения всех переменных текущей процедуры, а не только специально выбранных, как это происходит в окне **Watches**. Внешний же вид и структура обеих окон **Watches** и **Locals** одни и те же.

Окно *Immediate*

Окно **Immediate**, отображаемое на экране командой **View | Immediate Window**, предоставляет пользователю следующие возможности:

- набирать и вычислять отдельные инструкции VBA. Для этого достаточно в окне **Immediate** ввести соответствующую инструкцию и нажать клавишу <Enter>. Единственным ограничением на инструкцию является то, что она должна быть набрана в одну строку. Например,

```
s = 0: For i=1 to 5: s = s + i ^ 2: Next i: MsgBox s
```

- определять текущие значения переменных и свойств. Для этого в окне **Immediate** надо набрать вопросительный знак, имя переменной или свойства и нажать клавишу <Enter>. Например,

```
?x
```

- устанавливать новые текущие значения переменных. Для этого в окне **Immediate** надо набрать имя переменной, знак "равно" и новое значение переменной:

```
x = 15
```

- определять значения встроенных в VBA констант. Для этого в окне **Immediate** надо набрать вопросительный знак, имя константы и нажать клавишу <Enter>. Например,

```
? vbKeyReturn
```

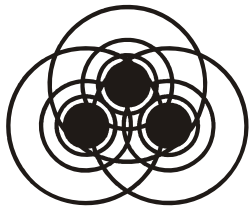
Программный способ вывода значений в окно *Immediate*

Существует также программный способ вывода значения свойств и переменных в диалоговое окно **Immediate** при помощи метода `Print` объекта `Debug`. Ниже приведен пример программного способа вывода значений переменных в окно **Immediate** (листинг 13.7).

Листинг 13.7. Программный способ вывода значений в окно Immediate

```
Dim n As Integer, Res As Integer
Randomize
For n = 1 To 10
   Res = Int(6 * Rnd()) + 1
   Debug.Print n, Res
Next
```

Глава 14



VBA и Win32 API

Win32 API (dynamic-link library, библиотека динамической компоновки) предоставляет в распоряжение разработчика богатую коллекцию функций и процедур, позволяющих решить разнообразные задачи. На данный момент использование Win32 API является стандартом для любой среды или языка программирования, это и понятно, как иначе писать программы для Windows? Вместе с тем пользоваться этим же API надо осторожно, ее реализации в версиях Windows отличаются вплоть до присутствия некоторых функций. API функции позволяют не только управлять как данным приложением, так и Windows, с их помощью возможно создавать различные спецэффекты, например, прозрачная форма или форма произвольной формы.

Первый пример

Для того чтобы использовать функции Win32 API, их необходимо объявить, используя `Declare`. Следующий код (листинг 14.1) демонстрирует вызов API функции `sndPlaySound32`, используемой для проигрывания Wav-файлов. Эта функция имеет два параметра, первый из которых задает имя файла, а второй — способ его проигрывания.

Листинг 14.1. Проигрывание Wav-файла

```
Declare Function sndPlaySound32 Lib "winmm.dll" Alias "sndPlaySoundA" _
       (ByVal lpszSoundName As String, ByVal uFlags As Long) As Long

Sub PlaySound(ByVal fileName As String)
   Call sndPlaySound32(Application.StartupPath + "\" + fileName, 0)
End Sub
```

Синтаксис инструкции `Declare` в общем случае имеет следующий вид и зависит от того, что объявляется — функция или процедура.

```
[Public или Private] Declare Sub имя_процедуры lib
   "имя_динамической_библиотеки" [Alias "псевдоним"] [(параметры)]
```

```
[Public или Private] Declare Function имя_процедуры Lib
   "имя_динамической_библиотеки" [Alias "псевдоним"] [(параметры)]
   [As тип]
```

Отсылка электронной почты и загрузка Web-страницы

Для отсылки электронной почты или загрузка Web-страницы можно воспользоваться функцией `ShellExecute`, которая открывает или печатает специфицированные файлы. Ее первым параметром является значение указателя того окна, в котором документ открывается. Второй параметр задает выполняемую операцию, третий идентифицирует файл, с которым производится работа, четвертый — передаваемые параметры, пятый — используемый по умолчанию каталог и шестой специфицирует то, как будет отображаться приложение при его открытии. Если функция `ShellExecute` открывает документ в новом окне, то в качестве значения ее первого параметра можно указать ссылку на рабочий стол, которая возвращается функцией `GetDesktopWindow`. В коде из листинга 14.2 процедура `SendMail` применяется для открытия окна MS Outlook для посылки электронной почты по адресу `agarnaev@rambler.ru`, а процедура `LoadWebPage` загружает в новое окно браузера Web-страницу `http://www.bhv.ru`.

Листинг 14.2. Отсылка электронной почты и загрузка Web-страницы

```
Const SW_SHOWNORMAL = 1

Private Declare Function ShellExecute Lib "shell32.dll" _
            Alias "ShellExecuteA" _
            (ByVal hwnd As Long, _
            ByVal lpOperation As String, _
            ByVal lpFile As String, _
            ByVal lpParameters As String, _
            ByVal lpDirectory As String, _
            ByVal nShowCmd As Long) As Long

Private Declare Function GetDesktopWindow Lib "user32" () As Long

Private Sub SendMail()
    Call ShellExecute(GetDesktopWindow, vbNullString, _
            "mailto:agarnaev@rambler.ru", _
            vbNullString, "c:\", SW_SHOWNORMAL)
```

```
End Sub


Sub LoadWebPage()
   Call ShellExecute(GetDesktopWindow, _
        "Open", "http://www.bhv.ru", _
         vbNullString, _
         "c:\", _
         SW_SHOWNORMAL)
End Sub
```

Определение и изменение разрешения экрана

При создании приложений, которые будут распространяться, важно уметь определять разрешение экрана пользовательского компьютера. Здесь на помощь приходит API функция `GetSystemMetrics`, имеющая единственный параметр, который и специфицирует используемую метрику. Данный код (листинг 14.3) демонстрирует возможность применения этой функции. При этом пользователь не только информируется о текущем разрешении экрана, но также ему предлагается средство его изменения.

Листинг 14.3. Определение и изменение разрешения экрана

```
Declare Function GetSystemMetrics _
    Lib "user32.dll" (ByVal nIndex As Long) As Long

Const SM_CXSCREEN = 0
Const SM_CYSCREEN = 1

Sub Get_Change_ScreenSize()
   Dim x As Long, y As Long, msg, answer As Integer
   x = GetSystemMetrics(SM_CXSCREEN)
   y = GetSystemMetrics(SM_CYSCREEN)
   msg = "Размеры экрана " & x & " x " & y & vbCrLf
   msg = msg & "Изменить их?"
   answer = MsgBox(msg, vbExclamation + vbYesNo, "Размеры экрана")
   If answer = vbYes Then
      Call Shell("rundll32.exe shell32.dll,Control_RunDLL desk.cpl,,3")
   End If
End Sub
```

Как открыть окно Проводника

Для открытия окна Проводника достаточно запустить на исполнение программу Explorer.exe, что можно сделать при помощи функции `WinExec`, как это показано в листинге 14.4. У этой функции имеются два параметра. Первый из них идентифицирует открываемое приложение, а второй — режим его выполнения.

Листинг 14.4. Как открыть окно Проводника

```
Private Declare Function WinExec Lib "kernel32" _
            (ByVal lpCmdLine As String, _
            ByVal nCmdShow As Long) As Long

Private Sub Startapplic()
   Call WinExec("Explorer.exe c:\", 1)
End Sub
```

Как отобразить окно свойств заданного файла

Для отображения окна свойств заданного файла надо воспользоваться функцией `ShellExecuteEx`, запускающей указанную программу или открывающей данный документ, например, как показано в листинге 14.5. Параметром этой функции является экземпляр структуры `SHELLEXECUTEINFO`, которая инкапсулирует в себе данные об исполняемом приложении.

Листинг 14.5. Как отобразить окно свойств заданного файла

```
Private Type SHELLEXECUTEINFO
   cbSize As Long
   fMask As Long
   hWnd As Long
   lpVerb As String
   lpFile As String
   lpParameters As String
   lpDirectory As String
   nShow As Long
   hInstApp As Long
   lpIDList As Long
```

```
   lpClass As String
   hkeyClass As Long
   dwHotKey As Long
   hIcon As Long
   hProcess As Long
End Type

Private Const SEE_MASK_INVOKEIDLIST = &HC
Private Const SEE_MASK_NOCLOSEPROCESS = &H40
Private Const SEE_MASK_FLAG_NO_UI = &H400

Private Declare Function ShellExecuteEx Lib "shell32" _
   Alias "ShellExecuteExA" (sei As SHELLEXECUTEINFO) As Long

Private Declare Function GetDesktopWindow Lib "user32" () As Long

Private Sub ShowPropertyWindow(strFilename As String, hWnd As Long)
   Dim lngReturn As Long
   Dim sei As SHELLEXECUTEINFO

   With sei
      .cbSize = Len(sei)
      .fMask = SEE_MASK_NOCLOSEPROCESS Or SEE_MASK_INVOKEIDLIST _
               Or SEE_MASK_FLAG_NO_UI
      .hWnd = hWnd
      .lpVerb = "properties"
      .lpFile = strFilename
      .lpParameters = vbNullChar
      .lpDirectory = vbNullChar
      .nShow = 0
      .hInstApp = 0
      .lpIDList = 0
   End With
   lngReturn = ShellExecuteEx(sei)
End Sub

Private Sub DemoShowPropertyWindow()
   ShowPropertyWindow "c:\test.txt", GetDesktopWindow
End Sub
```

Как загрузить Web-страницу, адрес которой набран в окне ввода

Для того чтобы загрузить Web-страницу, адрес которой набран в окне ввода, достаточно воспользоваться API функцией `ShellExecute`, выполняющей указанное приложение, например, как продемонстрировано в листинге 14.6. Если в качестве значения параметра `lpFile` этой функции указать URL Web-страницы, то эта страница загрузится в окно браузера.

Листинг 14.6. Загрузка Web-страницы, адрес которой набран в окне ввода

```
Private Declare Function ShellExecute Lib "shell32.dll" _
      Alias "ShellExecuteA" (ByVal hWnd As Long, _
      ByVal lpOperation As String, _
      ByVal lpFile As String, ByVal _
      lpParameters As String, ByVal lpDirectory As String, _
      ByVal nShowCmd As Long) As Long


Public Sub LoadWebSite()
   Dim DefaultURL As String
   Dim InputURL As String
   On Error GoTo ErrorHandler
   If Len(Selection.Text) > 0 Then
      DefaultURL = Selection.Text
   Else
      DefaultURL = "http://www.microsoft.com"
   End If
   InputURL = InputBox("Enter the URL for your default browser", _
            "              Jump to the Internet", DefaultURL)
   If InputURL = Cancel Then GoTo ErrorHandler
   ShellExecute 0&, vbNullString, InputURL, vbNullString, _
                vbNullString, SW_SHOWNORMAL
ErrorHandler:
End Sub
```

Полупрозрачная форма

Функция `SetLayeredWindowAttributes` задает полупрозрачную форму. Первый ее параметр задает указатель на форму, второй — индекс окна, а третий — коэффициент непрозрачности, которым может быть целое число из

диапазона от 0 до 255. Для получения указателя на форму надо воспользоваться функцией `GetActiveWindow`. Кроме того, нам потребуются две функции `GetWindowLong` и `SetWindowLong`, возвращающие и устанавливающие атрибуты окна. В следующем примере (листинг 14.7) на форме расположен счетчик (рис. 14.1). Счетчик управляет непрозрачностью формы, причем информация о текущей степени непрозрачности отображается в заголовке формы.

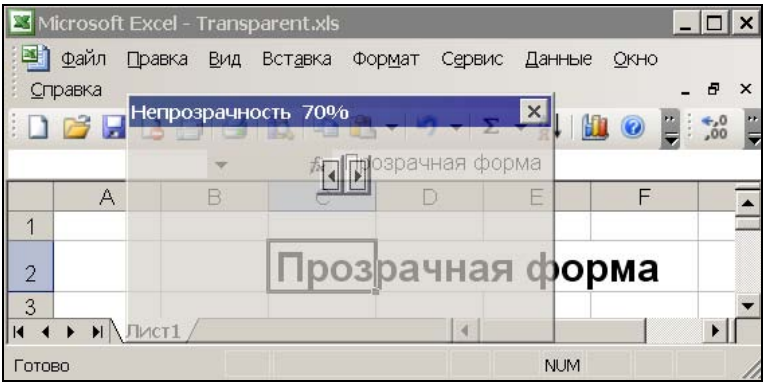


Рис. 14.1. Полупрозрачная форма

Листинг 14.7. Как создать полупрозрачную форму

```
Private Const WS_EX_LAYERED = &H80000
Private Const LWA_COLORKEY = &H1
Private Const LWA_ALPHA = &H2
Private Const GWL_EXSTYLE = &HFFEC
Dim hWnd As Long

Private Declare Function GetActiveWindow Lib "user32"() As Long

Private Declare Function SetWindowLong Lib "user32" _
    Alias "SetWindowLongA" ( _
    ByVal hWnd As Long, ByVal lngWinIdx As Long, _
    ByVal dwNewLong As Long) As Long

Private Declare Function GetWindowLong Lib "user32" _
    Alias "GetWindowLongA" ( _
    ByVal hWnd As Long, ByVal lngWinIdx As Long) As Long

Private Declare Function SetLayeredWindowAttributes Lib "user32"( _
```

```
    ByVal hWnd As Long, ByVal crKey As Integer, _
    ByVal bAlpha As Integer, ByVal dwFlags As Long) As Long

Private Sub SpinButton1_Change()
   SetOpacity SpinButton1.Value
End Sub

Private Sub UserForm_Activate()
   SpinButton1.Value = 50
End Sub

Private Sub SetOpacity(ByVal op As Integer)
   Dim lngWinIdx As Long
   hWnd = GetActiveWindow
   lngWinIdx = GetWindowLong(hWnd, GWL_EXSTYLE)
   lngWinIdx = lngWinIdx Or WS_EX_LAYERED
   SetWindowLong hWnd, GWL_EXSTYLE, lngWinIdx
   SetLayeredWindowAttributes hWnd, 0, (255 * op / 100), LWA_ALPHA
   Me.Caption = "Непрозрачность  " & op & "%"
End Sub
```

Форма произвольной формы

Пользовательские формы могут быть не только прозрачными, но и произвольной формы. Для этого достаточно при помощи API функций создать искомый контур, а затем API функцией `SetWindowRgn` отрезать от формы все лишнее. Рисовать контур можно API функциями:

- `CreateRectRgn` — создает прямоугольник;
- `CreateRoundRectRgn` — создает прямоугольник с закругленными углами;
- `CreateEllipticRgn` — создает эллипс;
- `CreatePolygonRgn` — создает многоугольник.

Функция `CombineRgn` конструирует на основе двух областей третью область. При комбинировании этих областей используется операция, заданная третьим параметром данной функции. Функция `DeleteObject` удаляет из памяти данный объект.

В приводимом ниже примере демонстрируется данный подход для создания формы в виде восьмерки (рис. 14.2). Расположите на форме кнопку и, используя окно **Properties,** установите значения ее свойств `Name` и `Caption` равными `cmdExit` и `Exit`. В модуле формы наберите код из листинга 14.8, *а*, а в стандартном модуле — код из листинга 14.8, *б*, в котором объявляются

все упомянутые выше API функции. Кнопка **Exit** используется для закрытия приложения, и, кроме того, в коде надо явным образом перепрограммировать перемещение формы, т. к. у нее отсутствует граница, за которую и осуществлялась буксировка формы.

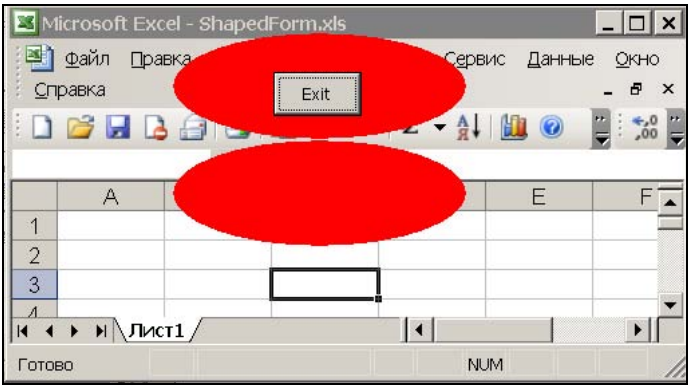


Рис. 14.2. Форма произвольной формы

Листинг 14.8, *а*. Форма произвольной формы. Модуль формы

```
Private Sub UserForm_Activate()
   Me.BackColor = vbRed
   Me.Repaint
   Dim rgn1 As Long, rgn2 As Long, rgn As Long
   rgn1 = CreateEllipticRgn(20, 40, 240, 120)
   rgn2 = CreateEllipticRgn(20, 120, 240, 200)
   rgn = CreateEllipticRgn(0, 0, 0, 0)
   CombineRgn rgn, rgn1, rgn2, RGN_OR
   Dim hndl As Long
   hndl = GetActiveWindow
   SetWindowRgn hndl, rgn, True
   DeleteObject rgn
   cmdExit.Top = 25
   cmdExit.Left = 55
   cmdExit.Width = 40
   cmdExit.Height = 20
End Sub

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

Листинг 14.8, *б*. Форма произвольной формы. Стандартный модуль

```
Public Const RGN_COPY = 5
Public Const RGN_AND = 1
Public Const RGN_XOR = 3
Public Const RGN_OR = 2
Public Const RGN_MIN = RGN_AND
Public Const RGN_MAX = RGN_COPY
Public Const RGN_DIFF = 4

Private Type POINTAPI
    x As Long
    y As Long
End Type

Public Declare Function GetActiveWindow Lib "user32" () As Long
Public Declare Function DeleteObject Lib "gdi32" _
       (ByVal hObject As Long) As Long
Public Declare Function CreateRectRgn Lib "gdi32" _
       (ByVal X1 As Long, ByVal Y1 As Long, _
        ByVal X2 As Long, ByVal Y2 As Long) As Long
Public Declare Function CreateRoundRectRgn Lib "gdi32" _
       (ByVal X1 As Long, ByVal Y1 As Long, _
        ByVal X2 As Long, ByVal Y2 As Long, _
        ByVal X3 As Long, ByVal Y3 As Long) As Long
Public Declare Function CreateEllipticRgn Lib "gdi32" _
       (ByVal X1 As Long, ByVal Y1 As Long, _
        ByVal X2 As Long, ByVal Y2 As Long) As Long
Public Declare Function CreatePolygonRgn Lib "gdi32" _
       (lpPoint As POINTAPI, ByVal nCount As Long, _
        ByVal nPolyFillMode As Long) As Long
Public Declare Function CombineRgn Lib "gdi32" _
       (ByVal hDestRgn As Long, ByVal hSrcRgn1 As Long, _
        ByVal hSrcRgn2 As Long, ByVal nCombineMode As Long) As Long
Public Declare Function SetWindowRgn Lib "user32" _
       (ByVal hWnd As Long, ByVal hRgn As Long, _
        ByVal bRedraw As Long) As Long
```

Форма без заголовка

Функция `DrawMenuBar` рисует строку меню окна. Эта функция используется, если строка меню изменялась после загрузки формы. С помощью данной функции можно, в частности, создать окно без заголовка, как это делается в листинге 14.9.

Листинг 14.9. Форма без заголовка

```
Private Const GWL_STYLE = -16
Private Const WS_CAPTION = &HC00000
Private Declare Function GetWindowLong Lib "user32" _
    Alias "GetWindowLongA" ( _
    ByVal hWnd As Long, _
    ByVal nIndex As Long) As Long
Private Declare Function SetWindowLong Lib "user32" _
    Alias "SetWindowLongA" ( _
    ByVal hWnd As Long, _
    ByVal nIndex As Long, _
    ByVal dwNewLong As Long) As Long
Private Declare Function DrawMenuBar Lib "user32" ( _
    ByVal hWnd As Long) As Long

Private Declare Function FindWindowA Lib "user32" ( _
    ByVal lpClassName As String, _
    ByVal lpWindowName As String) As Long

Private Sub UserForm_Initialize()
    Dim lngWindow As Long
    Dim lFrmHdl As Long
    lFrmHdl = FindWindowA(vbNullString, Me.Caption)
    lngWindow = GetWindowLong(lFrmHdl, GWL_STYLE)
    lngWindow = lngWindow And (Not WS_CAPTION)
    Call SetWindowLong(lFrmHdl, GWL_STYLE, lngWindow)
    Call DrawMenuBar(lFrmHdl)
End Sub

Private Sub UserForm_DblClick(ByVal Cancel As MSForms.ReturnBoolean)
   Unload Me
End Sub
```

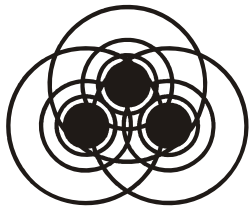
Заголовок активного окна

Функция `CaptionOfActiveWindow`, описанная в данном примере (листинг 14.10), возвращает заголовок активного окна. Она использует две API функции: функцию `GetActiveWindow`, возвращающую ссылку на активное окно, и функцию `GetWindowText`, возвращающую заголовок окна с указанной ссылкой.

Листинг 14.10. Заголовок активного окна

```
Declare Function GetActiveWindow Lib "user32" () As Long
Declare Function GetWindowText Lib "user32" _
   Alias "GetWindowTextA" (ByVal Hwnd As Long, _
   ByVal lpString As String, ByVal cch As Long) As Long

Function CaptionOfActiveWindow() As String
   Dim strCaption As String
   Dim lngLen As Long
   strCaption = String$(255, vbNullChar)
   lngLen = Len(strCaption)
   If GetWindowText(GetActiveWindow, strCaption, lngLen) > 0 Then
      CaptionOfActiveWindow = strCaption
   End If
End Function
```

Глава 15



VBA и Word

VBA в Word используются для автоматизации работы с документом, программирования пользовательских стилей. В данной главе сначала описывается, как можно создать шаблон документа с индивидуальным интерфейсом, как используются формы при работе со стандартными документами, а затем приводится список полезных макросов: от сбора статистических данных о документе до отображения информации о доле просмотренного документа в статусной строке.

Создание стилей

Стилем называется набор параметров форматирования, который применяется к тексту, таблицам и спискам, чтобы быстро изменить их внешний вид. Стили позволяют одним действием применить сразу всю группу атрибутов форматирования. Например, вместо форматирования названия в три приема, когда сначала задается размер 20 пунктов, затем полужирный шрифт Times New Roman и, наконец, выравнивание по центру, можно все это сделать одновременно, применив стиль **Заголовок 1**. Если же этот стиль заголовка вас не устраивает, например, требуется, чтобы высота была не 16, а 18 пунктов, шрифт не Times New Roman, а Arial Black причем красного цвета и, кроме того, заголовок имел рамку, то на основе стандартного стиля заголовка можно создать пользовательский, а затем его использовать.

Для создания нового стиля:

- выберите команду **Формат | Стили и форматирование**. В результате в области задач откроется раздел **Стили и форматирование**;
- нажмите кнопку **Создать стиль**. На экране отобразится окно **Создание стиля** (рис. 15.1);
- в поле имя стиля введите имя создаваемого стиля, например `MyHead`;
- в списке **Стиль** укажите тип создаваемого стиля: **Абзац**, **Знак**, **Таблица** или **Список**. В нашем случае выберете **Абзац**;
- в списке **Основан на стиле** выберите стиль, на основе которого создается ваш стиль. В данном случае выберите **Заголовок 1**;

- в списке **Стиль следующего абзаца** установите то, как будет выглядеть абзац, идущий за данным. В нашем случае выберите **Основной текст**;
- при помощи списка **Формат** можно задать индивидуальное форматирование для шрифта, абзаца, рамки, границы, нумерации, языка и сочетания клавиш. Установите, например, Arial Black красный шрифт высотой 18 пунктов, а рамку — в виде волнистой черты, расположенной только в нижней части области;
- флажок **Добавить в шаблон** позволяет добавить данный стиль в шаблон, а флажок **Обновлять автоматически** позволяет изменять стиль прямо из документа.

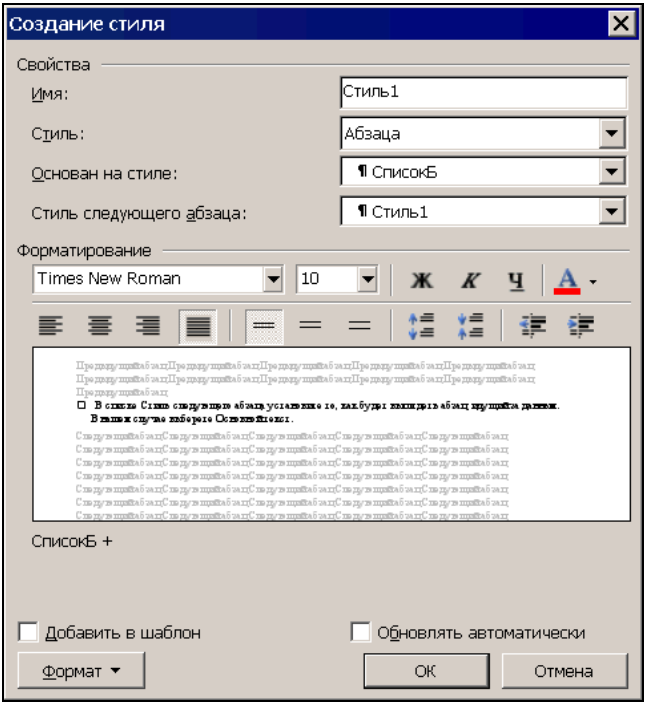


Рис. 15.1. Окно **Создание стиля**

Назначение стиля кнопке пользовательского меню

Итак, стиль создан. Теперь его надо будет назначить кнопке пользовательского меню. Для этого:

- выберите команду **Вид | Панели инструментов | Настройка**. На экране отобразится окно **Настройка**. На вкладке **Панели инструментов** нажмите

кнопку **Создать**. На экране отобразится окно **Создание панели инструментов** (рис. 15.2). В поле **Панель инструментов** введите имя панели, например `MyPanel`. В списке **Сделать панель доступной для** выберите ссылку на текущий документ, в данном случае **Документ 1**. Нажмите кнопку **OK**. Окно **Создание панели инструментов** закроется, а на экране появится пустая панель инструментов **MyPanel**;

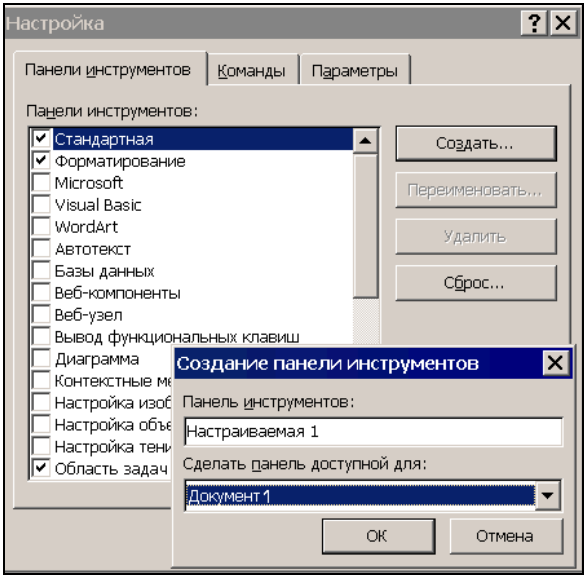


Рис. 15.2. Окна **Настройка** и **Создание панели инструментов**

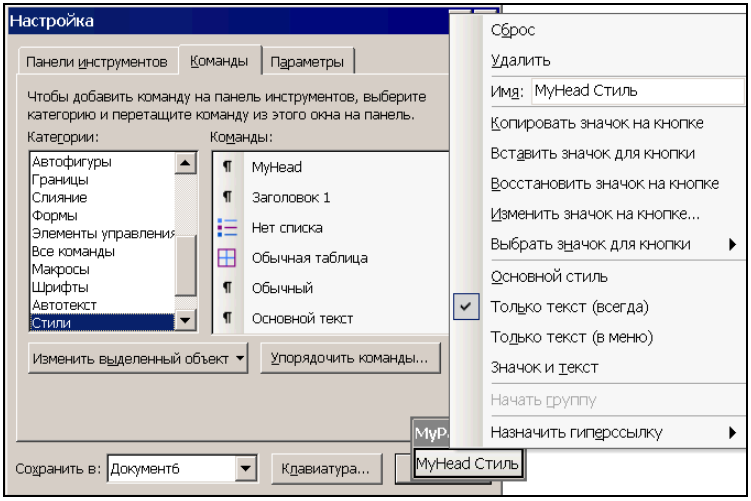


Рис. 15.3. Вкладка **Команды** окна **Настройка**, панель инструментов **MyPanel** с добавленной кнопкой и контекстным меню

- в списке **Категории** вкладки **Команды** окна **Настройка** выберите **Стили**. В списке **Команды** выберите **MyHead** и пробуксируйте этот значок на панель инструментов **MyPanel**. Итак, кнопка, задающая стиль, добавлена в панель инструментов;
- с помощью контекстного меню, отображаемого при нажатии правой клавишей мыши на созданной кнопке, можно изменить ее внешний вид, текст, установить флажок или связать с ней макрос (рис. 15.3). В данном случае ничего делать не будем, оставим все установки, назначенные по умолчанию. Просто нажмите кнопку **Закрыть**.

Добавление в пользовательское меню кнопки, которой назначен макрос

Добавим в построенную панель инструментов **MyHead** еще одну кнопку (например, **Об авторе**), которая запускает на выполнение простейший макрос, отображающий окно с информацией об авторе приложения.

- Для создания макроса надо перейти в редактор Visual Basic. Для этого нажмите комбинацию клавиш <Alt>+<F11>. Добавьте в проект модуль при помощи команды **Insert | Module** и в нем наберите следующий код (рис. 15.4).

```
Sub ShowAuthor()
   MsgBox "Программа была " & vbCr & _
          "создана мною"
End Sub
```

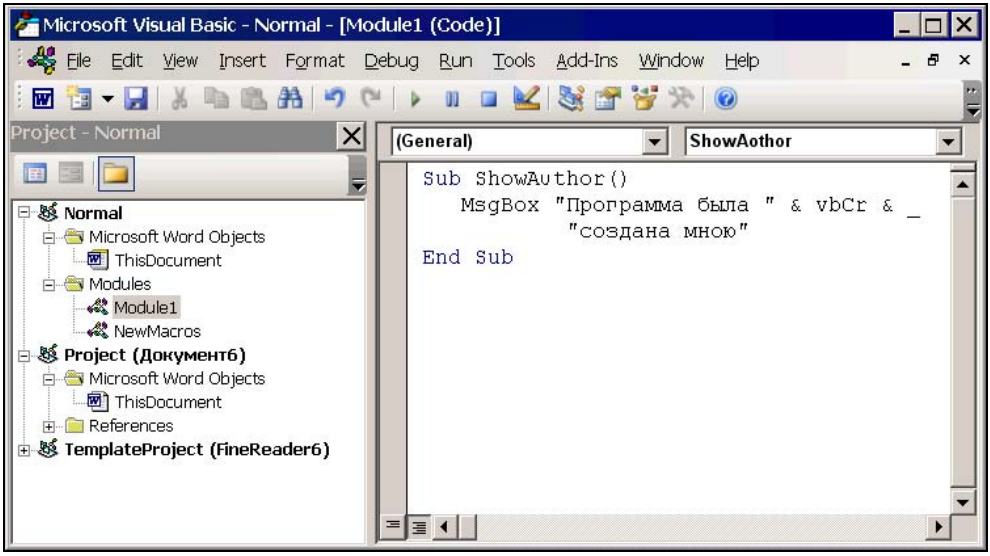


Рис. 15.4. Редактор Visual Basic с набранным в модуле кодом

- Для добавления в панель инструментов новой кнопки командой **Вид | Панели инструментов | Настройка** отобразите окно **Настройка**. В списке **Категории** вкладки **Команды** выберите **Макросы**, а в списке **Команды** выберите ссылку на макрос **ShowAuthor** и пробуксируйте этот значок на панель инструментов **MyPanel**. Итак, кнопка, активизирующая макрос, создана. С помощью контекстного меню установите у нее надпись **Об авторе** (рис. 15.5). Нажмите кнопку **Закрыть**. Панель инструментов готова.

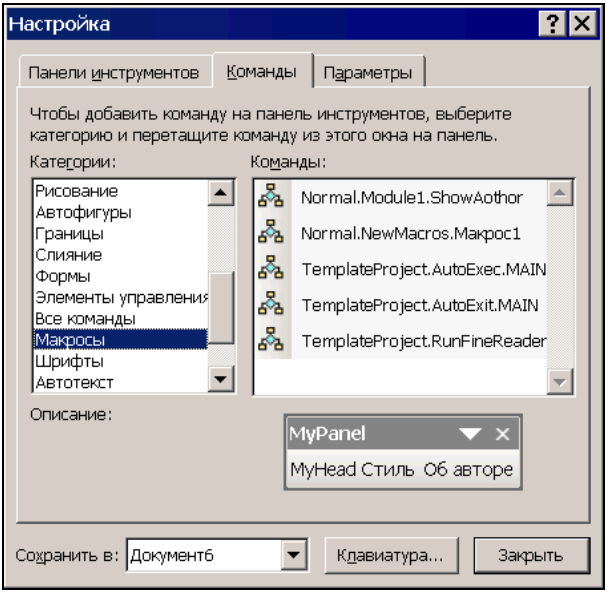


Рис. 15.5. Вкладка **Команды** окна **Настройка** и панель инструментов **MyPanel** с двумя кнопками

Программное задание стиля

Стиль можно сначала создать, а задать его уже потом в виде макроса при помощи свойства `Style` объекта `Selection` (листинг 15.1).

Листинг 15.1. Программное задание стиля

```
Sub SetStyleMyHead()
   Selection.Style = "MyHead"
End Sub
```

Создание шаблонов

При работе с нестандартными стилями удобным средством является шаблон. Можно создать собственные шаблоны для разных нужд. Затем, по мере необходимости, новые документы создаются уже на базе существующих шаблонов, которые унаследуют от них сконструированные стили. Например, для создания шаблона с нашей панелью инструментов **MyPanel**, позволяющей задавать стиль заголовка:

- выберите команду **Файл | Сохранить как**;
- в поле **Тип файла** выберите **Шаблон документа (*.dot)**;
- в поле **Имя файла** введите имя нового шаблона, например `My`, и нажмите кнопку **Сохранить**.

По умолчанию файл с шаблоном имеет расширение DOT и сохраняется в папке **Шаблоны**, которая спрятана глубоко в недрах Windows где-то по адресу **\Microsoft\Templates**.

Создание документа по шаблону

Для создания документа по шаблону:

- выберите команду **Файл | Создать**;
- в области задач **Создать документ** в группе **Шаблоны** выберите одну из ссылок на местоположение шаблона (**На моем компьютере**, **Шаблоны на узле Office Online** и **На моих Web-узлах**). Можно также произвести поиск шаблона в сети по имени, введя его в поле **Поиск в сети** и нажав кнопку **Найти**;
- если шаблон расположен на вашем компьютере, как в данной ситуации, то нажмите кнопку **На моем компьютере**. Отобразится окно **Шаблоны**. На вкладке **Общие** выберите ссылку на шаблон My.dot, убедитесь, что установлен переключатель **документ** в группе **Создать,** и нажмите кнопку **OK**.

Присоединение шаблона к существующему документу

Для присоединения шаблона к существующему документу:

- Выберите команду **Сервис | Шаблоны и настройки**;
- на вкладке **Шаблоны** в поле **Шаблон** документа при помощи кнопки **Присоединить** установите ссылку на искомый шаблон. Нажмите кнопку **OK**.

Вставка символа

Конечно, пользовательский интерфейс в Word может быть использован не только для задания стилей, но и для любых других задач, которые возникают у разработчика. Например, предположим, что вы часто набираете текст, в который надо добавлять специальные символы. Почему бы не создать форму, на которой собраны эти символы, и все что требуется — просто вызвать эту форму и выбрать соответствующий символ. Например, создадим форму для ввода символов " " и " " (рис. 15.6). Для этого достаточно на форме расположить две кнопки и в окне **Properties** установить у них значения свойства `Name` и `Caption` равными `cmdLeft`, `<--` и `cmdRight`, `-->`, а в модуле формы набрать следующий код (листинг 15.2).

Рис. 15.6. Окно **Символы**

Листинг 15.2. Вставка символов

```
Private Sub cmdLeft_Click()
   Selection.InsertSymbol CharacterNumber:=8592, _
                          Unicode:=True
End Sub

Private Sub cmdRight_Click()
   Selection.InsertSymbol CharacterNumber:=8594, _
                          Unicode:=True
End Sub
```

Заполнение письма

Формы можно использовать для заполнения писем по шаблону. Например, ваш банк рассылает уведомительные письма об остатке на счету. Все эти письма имеют одну и туже структуру: строка обращения, текст сообщения и подпись. Итак, первоначально создайте шаблон документа (рис. 15.7). В нем создайте незаполненную таблицу, состоящую из трех строк и одного столбца, куда данные и будут вводиться.

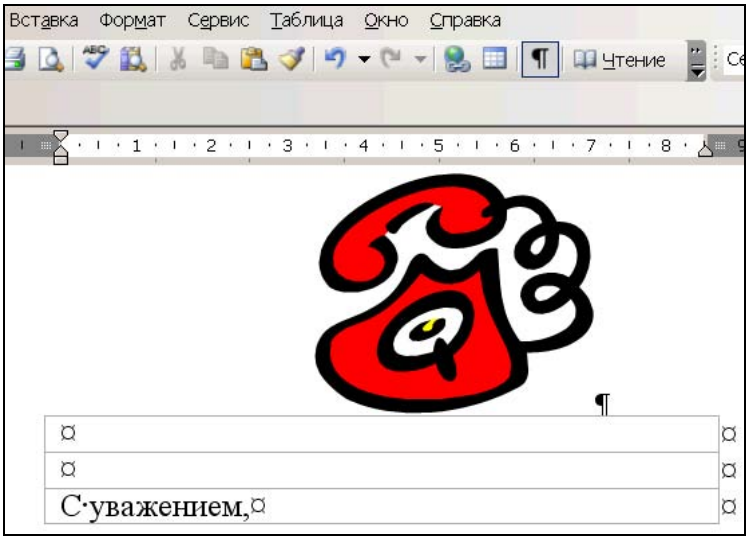


Рис. 15.7. Шаблон письма

Сконструируйте форму, на которой расположите рамку, два переключателя, поле, список и кнопку. Используя окно **Properties,** установите у элементов управления значения свойств, как показано в табл. 15.1.

Таблица 15.1. *Значения свойств, установленных в окне* ***Properties***

Элемент управления	Свойство	Значение
Рамка	Caption	Форма обращения
Переключатель	Name	optSir
	Caption	Господин
Переключатель	Name	optMiss
	Caption	Госпожа
Поле	Name	txtMessage
Список	Name	lstSignature
Кнопка	Name	OK
	Caption	cmdOK

В модуле формы наберите следующий код (листинг 15.3). Проект готов.

Листинг 15.3. Заполнение письма

```
Private Sub UserForm_Initialize()
   lstSignature.AddItem "Иванов"
   lstSignature.AddItem "Петров"
End Sub

Private Sub cmdOK_Click()
   If optSir.Value Then
      ActiveDocument.Tables(1).Cell(1, 1).Range.Text _
                  = "Уважаемый Господин"
   Else
      ActiveDocument.Tables(1).Cell(1, 1).Range.Text _
                  = "Уважаемая Госпожа"
   End If
   ActiveDocument.Tables(1).Cell(2, 1).Range.Text = txtMessage.Text
   ActiveDocument.Tables(1).Cell(3, 1).Range.InsertAfter Chr(13) _
                                      & lstSignature.Text
End Sub
```

Полезные макросы

Далее приведем ряд полезных макросов, используемых при работе с документами, от сбора статистических данных о документе, печати текущей строки до программного создания и удаления стилей.

Печать текущей страницы

Следующий макрос (листинг 15.4) применяется для печати только текущей страницы. Он использует метод `PrintOut` объекта `Application`. Параметр `Range` этого метода, установленный значению `wdPrintCurrentPage`, как раз и задает печатаемый диапазон.

Листинг 15.4. Печать текущей страницы

```
Sub PrintOutCurrentPage()
   Application.PrintOut FileName:="", _
                        Range:=wdPrintCurrentPage, _
                        Item:= wdPrintDocumentContent, _
                        Copies:=1, Pages:=""
End Sub
```

Получение статистических сведений о документе или выделенном фрагменте

Метод `ComputeStatistics` возвращает статистические сведения о документе или параграфе. Он имеет два параметра, первым из которых может быть одна из `WdStatistic` констант: `wdStatisticCharacters`, `wdStatisticCharactersWithSpaces`, `wdStatisticFarEastCharacters`, `wdStatisticLines`, `wdStatisticPages`, `wdStatisticParagraphs` и `wdStatisticWords`. Второй параметр является необязательным, принимает логические значения и определяет, надо ли включать в статистику данные из сносок.

Для получения статистических данных о документе можно также воспользоваться свойством `BuiltInDocumentProperties`. Следующие три примера (листинги 15.5, *а*, *б*, *в*) демонстрируют определение различных статистических сведений документа.

Листинг 15.5, *а*. Число слов и символов в первом параграфе указанного документа

```
Dim rgn As Range, nWord As Long, nChar As Long
Set rgn = Documents("Отчет.doc").Paragraphs(1).Range
nWord = rgn.ComputeStatistics(Statistic:=wdStatisticWords)
nChar = rgn.ComputeStatistics(Statistic:=wdStatisticCharacters)
MsgBox "Первый параграф содержит " & nWord _
 & " слов и " & nChar & " символов."
```

Листинг 15.5, *б*. Число слов в активном документе

```
MsgBox("Число слов в активном документе: " & _
       ActiveDocument.ComputeStatistics( _
       Statistic:=wdStatisticWords, _
       IncludeFootnotesAndEndnotes:=True)
```

Листинг 15.5, *в*. Определение числа страниц и слов в активном документе свойством `BuiltInDocumentProperties`

```
Function GetNumberOfPages() As Long
   ActiveDocument.Repaginate
   GetNumberOfPages = _
       ActiveDocument.BuiltInDocumentProperties(wdPropertyPages)
```

```
End Function

Function GetNumberOfWords() As Long
   GetNumberOfWords = _
        ActiveDocument.BuiltInDocumentProperties(wdPropertyWords)
End Function
```

Вставка фонового рисунка

Вставка фонового рисунка требует некоторых усилий. Для ускорения процесса можно воспользоваться следующей процедурой (листинг 15.6), которая с помощью окна **Открыть документ** позволяет отобрать и вставить фоновый рисунок в активный документ. Данные о фоне документа инкапсулируются в объекте `Background`.

Листинг 15.6. Вставка фонового рисунка

```
Sub SelectBackground()
   Dim dlg As Dialog
   Set dlg = Dialogs(wdDialogInsertPicture)
   If dlg.Display() Then
      ActiveDocument.ActiveWindow.View.Type = wdWebView
      ActiveDocument.Background.Fill.UserPicture PictureFile:= dlg.Name
   End If
End Sub
```

Двухстраничный и одностраничный просмотр документа

Следующие две процедуры (листинг 15.7) задают двухстраничный и одностраничный просмотр документа. Для обеспечения двухстраничного просмотра необходимо режим просмотра (значение свойства `Type` объекта `View`) установить равным `wdPrintView`, а затем при помощи свойства `PageColumns` объекта `Zoom` указать требуемое число страниц. Установка одностраничного просмотра осуществляется также, но в конце надо перейти в режим нормального просмотра документа.

Листинг 15.7. Просмотр документа

```
Sub TwoPagesView()
   ActiveWindow.View.Type = wdPrintView
   ActiveWindow.View.Zoom.PageColumns = 2
```

```
End Sub
Sub OnePagesView()
   ActiveWindow.View.Type = wdPrintView
   ActiveWindow.View.Zoom.PageColumns = 1
   ActiveWindow.View.Type = wdNormalView
End Sub

Sub StyleKiller(stylename As String)
  On Error Resume Next
  ActiveDocument.Styles(stylename).Delete
End Sub
```

Поиск текста в выделенном фрагменте

Следующий код при помощи метода `Find` позволяет произвести поиск указанной строки в выделенном фрагменте текста (листинг 15.8).

Листинг 15.8. Поиск текста в выделенном фрагменте

```
Sub SearchText(str As String)
   Selection.Find.ClearFormatting
   Selection.Find.Replacement.ClearFormatting
   With Selection.Find
      .Text = str
       .MatchWildcards = False
       .Replacement.Text = ""
       .Wrap = wdFindStop
       .Forward = True
   End With
   Selection.Find.Execute
End Sub
```

Окраска всех вхождений данного слова в документ в заданный цвет

Текст в документе можно не только искать, но и выполнять над ним полезные действия. Например, бывает необходимо выделить каким-то цветом все вхождения указанного слова в документ. Как это сделать? Очень просто, надо найденный текст заменить им же самим, но имеющим теперь уже указанный цвет. Следующий код (листинг 15.9), например, окрашивает все слова Microsoft в активном документе в красный цвет.

Листинг 15.9. Выделение цветом всех вхождений данного слова

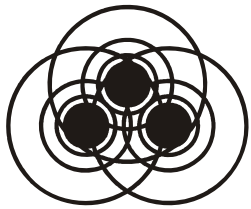
```
Sub TestColorWords()
     ColorWords "Microsoft", vbRed
End Sub

Sub ColorWords(ByVal strText As String, _
   ByVal MyColor As Long)

  With ActiveDocument.Content.Find
    .ClearFormatting
    With .Replacement
        .ClearFormatting
        .Font.Color = MyColor
    End With
    .Execute FindText:=strText, ReplaceWith:=strText, _
    Format:=True, Replace:=wdReplaceAll
End With
```

Отображение в строке состояния информации об относительном объеме просмотренного документа

Следующий код позволяет отобразить в строке состояния информацию об относительном объеме просмотренного документа.

```
Application.StatusBar = _
              Int(100 * (Selection.Start / ActiveDocument.Range.End)) _
              & "% просмотрено"
```

Глава 16



VBA и Automation

Средство *Automation*

Automation — это одно из наиболее важных средств технологии ActiveX, позволяющее программно управлять объектами из других приложений. Более ранняя технология называлась *Динамическим обменом данных* (DDE — Dynamic Data Exchange). Приложение поддерживает Automation одним из двух способов:

1. Приложение-источник (или *объект,* или *сервер Automation,* или *объект ActiveX*) — это приложение, объекты которого используются другим приложением или средствами программирования через интерфейс.
2. Приложение-приемник (или *контролер* или *клиент Automation*) — это приложение, которое управляет объектами приложения-источника.

> **Примечание**
>
> Некоторые приложения могут быть только приложениями-источниками, либо только приложениями- приемниками, но есть и такие, которые могут выступать и в той, и другой роли (например, Excel, Access).

Для программного управления объектом Automation надо:

1. Создать переменную, представляющую собой объект.
2. Использовать эту переменную для доступа к объектам, находящимся в приложениях-источниках. Объекты приложения источника образуют библиотеку *объектов серверов.*
3. По завершению работы с объектом, присвоить переменной значение `Nothing`, освобождая переменную.

Программные идентификаторы приложений-серверов *Automation*

Для идентификации приложений в технологии Automation используются программные идентификаторы, которые иногда обозначаются ProgID.

Обратите внимание на то, что если используется программный идентификатор без указания версии, то объект создается на основе наиболее современной установленной версии программы. В общем случае идентификатор ProgID имеет следующий синтаксис:

Appname.ObjectType

Здесь *Appname* — имя приложения-сервера, а *ObjectType* — тип или класс объектов. Например, для MS Excel этот идентификатор имеет значение `Excel.Application`, а MS Word — `Word.Application`.

Функции доступа к объектам *Automation*

Для доступа к объектам Automation приложения-сервера используются две функции `CreateObject` и `GetObject`.

Функция `CreateObject` создает и возвращает ссылку на объект ActiveX.

`CreateObject(`*Class*`,[`*Servername*`])`, где:

- *Class* — имя объекта Automation;
- *Servername* — параметр используется только при создании объекта Automation в сети и устанавливает имя сервера, где будет создан объект Automation.

Функция `GetObject` создает и возвращает ссылку на объект ActiveX, сохраненный в файле.

`GetObject([`*Pathname*`] [, `*Class*`])`, где:

- *Pathname* — полное имя файла; если параметр опущен, то необходимо указать значение параметра *Class*;
- *Class* — имя объекта Automation.

Функция `GetObject` подобна функции `CreateObject`. Но есть и некоторое различие между ними. Функцию `GetObject` можно использовать для доступа к существующим документам, хранящимся в файлах. Функцию `GetObject` можно также использовать и для доступа к объекту `Application` любого уже запущенного приложения Microsoft Office. Для этого надо вызвать функцию `GetObject` без первого параметра. Этот способ доступа к объекту `Application` любого уже запущенного приложения MS Office применяется, когда нет необходимости в запуске еще одного экземпляра приложения.

Например, в листинге 16.1, прежде чем получить доступ к MS Word сначала проверяется, не открыто ли это приложение. Если оно открыто, то используется работающее приложение, а если нет, то приложение MS Word запускается в работу.

Листинг 16.1. Загрузка MS Word при помощи функций GetObject и CreateObject

```
Sub Demo()
   Dim objWord As Object
   On Error GoTo APPLICATION_CREATE
   Set objWord = GetObject(, "Word.Application")
   GoTo SKIP_APPLICATION_CREATE
APPLICATION_CREATE:
   Set objWord = CreateObject("Word.Application")
   MsgBox "Открытие приложения, если это необходимо"
SKIP_APPLICATION_CREATE:
   MsgBox "Работа с приложением"
End Sub
```

Позднее и раннее связывание

Позднее связывание происходит, когда тип переменной, которая будет представлять собой объект Automation, имеет тип `Object`. Тип `Object` позволяет создавать объекты любой природы. В этом смысле он подобен типу `Variant`. Такая чрезмерная общность определения переменной понижает производительность приложения. Для достижения наилучшей производительности приложения необходимо определить конкретный тип для переменной, которая будет представлять собой объект Automation. Например, если используется MS Excel, то надо установить тип переменной `Excel.Application`.

Второй подход называется *ранним связыванием* и происходит на этапе компиляции. При раннем связывании, чтобы определить переменную определенного класса, перед написанием кода необходимо сослаться на библиотеку объектов серверов Automation. Для этого в редакторе VBA выберите команду **Tools | References**. В появившемся диалоговом окне **References** в списке **Available References** установите флажок **Microsoft Excel 11.0 Object Library**, при этом можно воспользоваться кнопкой **Browse**.

Выполнив предварительные действия, можно перейти к заданию нового экземпляра объекта Automation при помощи ключевого слова `New`. Например,

```
Dim objExcel As New Excel.Application
```

Открытие документа Word

Следующий пример (листинг 16.2) демонстрирует использование техники раннего связывания для открытия существующего документа Word (в данном случае C:\Test.doc), его визуализации, сообщения об этом пользователю, а затем его закрытия без сохранения. Для тестирования кода его можно

расположить в любом офисном приложении, например в модуле MS Excel. При этом не забудьте указать ссылку на библиотеку объектов **Microsoft Word 11.0 Object Library**.

Листинг 16.2. Открытие документа

```
Sub OpenWordDocument()
   Dim objWord As Word.Application
   Dim objDoc As Word.Document
   Dim FileName As String
   FileName = "C:\Test.doc"
   Set objWord = New Word.Application
   Set objDoc = objWord.Documents.Open(FileName)
   objWord.Visible = True
   MsgBox objDoc.Path & "\" & objDoc.Name
   objDoc.Close False
   objWord.Quit
   Set objDoc = Nothing
   Set appWord = Nothing
End Sub
```

Создание документа Word

В данном примере (листинг 16.3) не только создается новый документ Word, но в него также вводится отформатированное приветствие Hello, Automation World! При этом окно MS Word отображается в развернутом состоянии.

Листинг 16.3. Создание документа

```
Sub CreateWordDocument()
   On Error GoTo errorHandler
   Dim objWord As Word.Application
   Dim objDoc As Word.Document
   Dim objRgn As Word.Range
   Set objWord = New Word.Application
   With objWord
      .Visible = True
      .WindowState = wdWindowStateMaximize
   End With
   Set objDoc = objWord.Documents.Add
   Set objRgn = objDoc.Words(1)
```

```
   With objRgn
        .Text = "Hello, Automation World!"
        .Font.Name = "Comic Sans MS"
        .Font.Size = 12
        .Font.ColorIndex = wdBlue
        .Bold = True
   End With
errorHandler:
   Set objWord = Nothing
   Set objDoc = Nothing
   Set objRgn = Nothing
End Sub
```

Создание отчета Word на основе данных, хранимых на рабочем листе

Рассмотрим пример (листинг 16.4) создания отчета Word на основе данных, хранимых на рабочем листе. Итак, имеется рабочий лист **Отчет**, на котором расположена отчетная таблица и построенная на ее основе диаграмма (рис. 16.1). Эти данные надо импортировать в новый документ Word, добавив

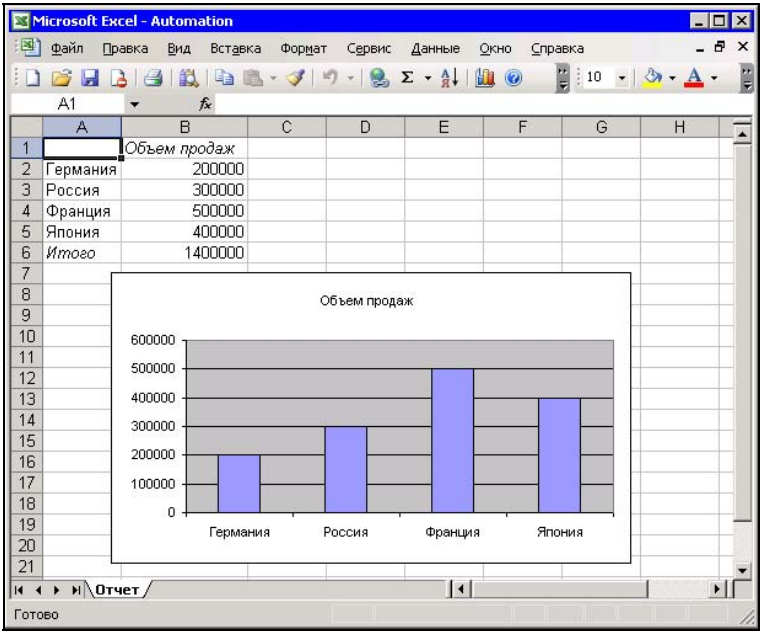


Рис. 16.1. Исходные данные, расположенные на рабочем листе **Отчет**

в него отформатированный заголовок, кроме того, с использованием автоформата, отформатировать саму таблицу (рис. 16.2). Все эти действия как раз и осуществляет приводимый ниже код. При реализации этого примера не забудьте указать ссылку на библиотеку объектов **Microsoft Word 11.0 Object Library**.

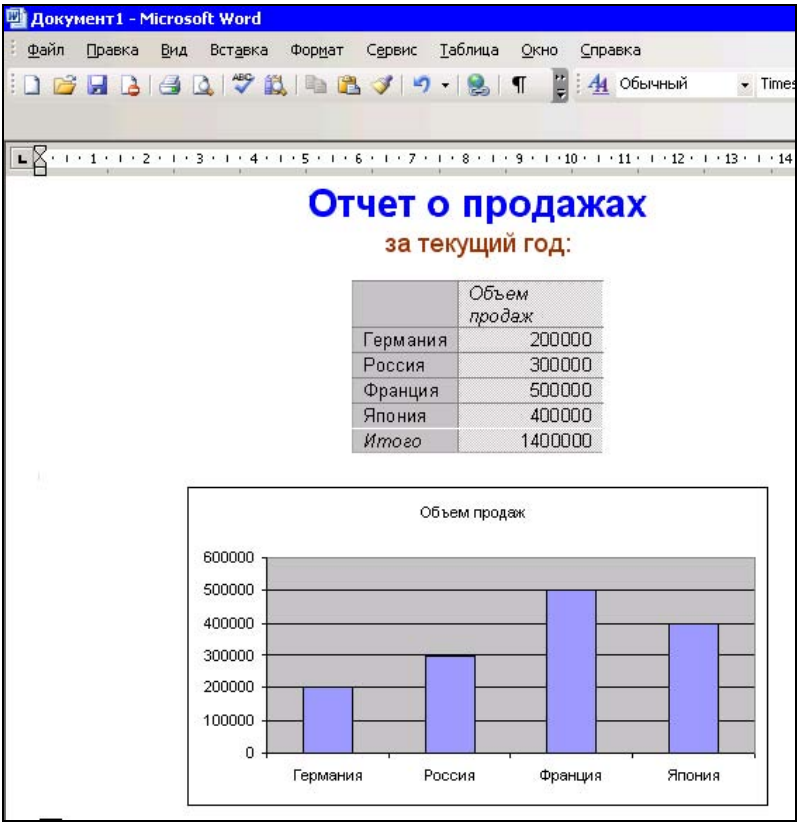


Рис. 16.2. Автоматически сгенерированный отчет

Листинг 16.4. Создание документа на основе данных из рабочей книги

```
Sub WordReportBuilder()
    Dim objWord As New Word.Application
    Dim objDoc As Word.Document
    With objWord
        .Visible = True
        .WindowState = wdWindowStateMaximize
        .Documents.Add
```

```
    Set objDoc = .ActiveDocument
End With

objDoc.ActiveWindow.View = wdNormalView
With objWord.Selection
    .InsertAfter "Отчет о продажах"
    .InsertParagraphAfter
    .InsertAfter "за текущий год:"
    .InsertParagraphAfter
    .MoveRight
End With

With objDoc.Paragraphs(1).Range
    .ParagraphFormat.Alignment = wdAlignParagraphCenter
    With .Font
        .Name = "Arial"
        .Size = 20
        .Bold = True
        .Color = wdColorBlue
        .Animation = wdAnimationSparkleText
    End With
End With
With objDoc.Paragraphs(2).Range
    With .ParagraphFormat
        .SpaceAfter = 12
        .Alignment = wdAlignParagraphCenter
    End With
    With .Font
        .Name = "Arial"
        .Size = 14
        .Color = wdColorBrown
    End With
End With

Worksheets("Отчет").Range("A1").CurrentRegion.Copy
With objWord.Selection
    .Paste
    .TypeParagraph
```

```
    End With

    objDoc.Tables(1).Select
    With objDoc.Tables(1)
        .Style = "Объемная таблица 3"
        .ApplyStyleHeadingRows = False
        .ApplyStyleLastRow = False
        .ApplyStyleFirstColumn = False
        .ApplyStyleLastColumn = False
    End With
    objDoc.Tables(1).Rows.Alignment = wdAlignRowCenter

    Dim n As Integer
    n = Worksheets("Отчет").Range("A1").CurrentRegion.Rows.Count + 1
    objWord.Selection.GoTo What:=wdGoToLine, Which:=wdGoToNext, Count:=n

    Worksheets("Отчет").ChartObjects(1).Copy
    With objWord
        .Selection.PasteSpecial Link:=False, _
            DataType:=wdPasteMetafilePicture, _
            Placement:=wdInLine, DisplayAsIcon:=False
        .Selection.ParagraphFormat.Alignment = _
            wdAlignParagraphCenter
        .Selection.GoTo What:=wdGoToLine, Which:=wdGoToAbsolute, _
            Count:=1
    End With
End Sub
```

Создание презентации на основе данных, хранимых на рабочем листе *MS Excel*

В следующем примере (листинг 16.5) на основе таблицы и диаграммы, расположенной на рабочем листе **Отчет** в PowerPoint создается презентация, состоящая из четырех слайдов, с автоматической их прокруткой (рис. 16.3). При реализации этого примера не забудьте указать ссылку на библиотеку объектов **Microsoft PowerPoint 11.0 Object Library**.

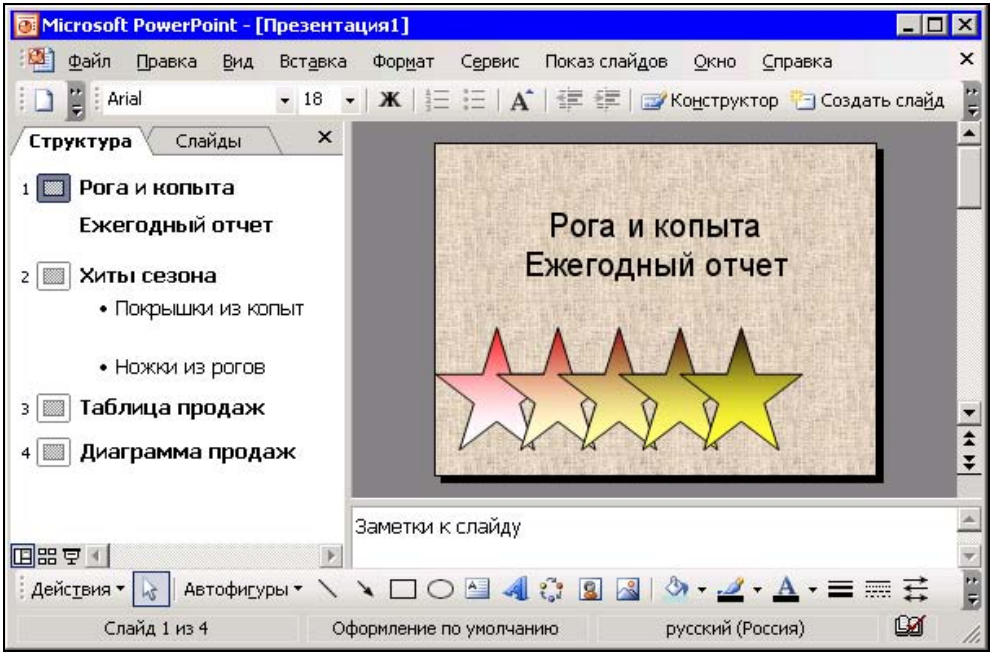


Рис. 16.3. Автоматически сгенерированная презентация

Листинг 16.5. Создание презентации на основе данных из рабочей книги

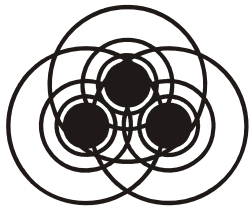
```
Sub BuilderPowerPointPresentation()
   Dim objPPoint As New PowerPoint.Application
   Dim objPres As PowerPoint.Presentation
   Dim objSlide(3) As PowerPoint.Slide
   Dim objShape(4) As PowerPoint.Shape
   Dim SlideNum As Integer

   With objPPoint
      .Visible = True
      .WindowState = ppWindowMaximized
      Set objPres = objPPoint.Presentations.Add
   End With

   Set objSlide(0) = objPres.Slides.Add(1, ppLayoutTitleOnly)
   objPres.SlideMaster.Background.Fill.PresetTextured msoTextureCanvas

   With objSlide(0)
```

```
    With .Shapes(1)
        .Top = 120
        With .TextFrame.TextRange
            .Text = "Рога и копыта" & vbCr & "Ежегодный отчет"
            .Font.Size = 54
        End With
        With .AnimationSettings
            .EntryEffect = ppEffectCheckerboardDown
            .AdvanceMode = ppAdvanceOnTime
        End With
    End With

    With objSlide(0).Shapes
        For i = 0 To 4
            Set objShape(i) = .AddShape( _
                        msoShape5pointStar, 100 * i, 300, 200, 200)
            With objShape(i).Fill
                .ForeColor.RGB = RGB(255 - 50 * i, 0, 0)
                .BackColor.RGB = RGB(255, 255, 255 - 50 * i)
                .TwoColorGradient Style:=msoGradientHorizontal, _
                                  Variant:=1
            End With
            With objShape(i).AnimationSettings
                .EntryEffect = ppEffectRandom
                .AdvanceMode = ppAdvanceOnTime
            End With
        Next
    End With
    With .SlideShowTransition
        .AdvanceOnTime = True
        .AdvanceTime = 1
        .EntryEffect = ppEffectRandom
    End With
End With

SlideNum = objPres.Slides.Count + 1
Set objSlide(1) = objPres.Slides.Add(SlideNum, ppLayoutText)
objPPoint.ActiveWindow.View.GotoSlide SlideNum

With objSlide(1)
```

```
    .Background.Fill.PresetTextured msoTextureBouquet
    .Shapes(1).TextFrame.TextRange.Text = "Хиты сезона"
    With .Shapes(2)
       .TextFrame.TextRange.Text = "Покрышки из копыт" & _
                  vbCr & vbCr & _
                  "Ножки из рогов"
       With .AnimationSettings
          .EntryEffect = ppEffectFlyFromTop
          .TextUnitEffect = ppAnimateByWord
          .TextLevelEffect = ppAnimateByFirstLevel
          .AdvanceMode = ppAdvanceOnTime
      End With
    End With

    With .SlideShowTransition
       .AdvanceOnTime = True
       .AdvanceTime = 1
       .EntryEffect = ppEffectRandom
    End With
End With


SlideNum = objPres.Slides.Count + 1
Set objSlide(2) = objPres.Slides.Add(SlideNum, ppLayoutTitleOnly)
objPPoint.ActiveWindow.View.GotoSlide SlideNum
With objSlide(2)
    .Shapes(1).TextFrame.TextRange.Text = "Таблица продаж"
    With .SlideShowTransition
       .AdvanceOnTime = True
       .AdvanceTime = 1
       .EntryEffect = ppEffectRandom
    End With
End With

Worksheets("Отчет").Range("A1").CurrentRegion.Copy
objPPoint.ActiveWindow.View.Paste

SlideNum = objPres.Slides.Count + 1
Set objSlide(3) = objPres.Slides.Add(SlideNum, ppLayoutTitleOnly)
```

```
   objPPoint.ActiveWindow.View.GotoSlide SlideNum
   With objSlide(3)
      .Shapes(1).TextFrame.TextRange.Text = "Диаграмма продаж"
      With .SlideShowTransition
         .AdvanceOnTime = True
         .AdvanceTime = 1
         .EntryEffect = ppEffectRandom
      End With
   End With
   Worksheets("Отчет").ChartObjects(1).Copy
   objPPoint.ActiveWindow.View.Paste

   objPPoint.ActiveWindow.View.GotoSlide 1
   With objPres.SlideShowSettings
      .StartingSlide = 1
      .EndingSlide = 4
      .AdvanceMode = ppSlideShowUseSlideTimings
      .Run
   End With
End Sub
```

Глава 17



ADO (ActiveX Data Objects)

Технология ADO (ActiveX Data Objects) предоставляет разработчику универсальный метод извлечения и изменения данных из разнообразных источников, которые не ограничиваются только традиционными базами данных. Ими могут быть и файловая система, и Интернет. Рука об руку с ADO идет OLE DB — набор интерфейсов COM (Component Object Model, компонентная модель объектов), используемых для извлечения данных. OLE DB предоставляет низкоуровневое подключение к данным через интерфейс COM, в то время как ADO предоставляет модель объектов, которая упрощает процесс использования OLE DB и написания кода. О работе с базами данных в MS Excel при помощи технологии ODBC рассказано в моей книге "Excel, VBA, Internet в экономике и финансах", опубликованной издательством BHV-Петербург.

Технология ADO позволяет решать следующие задачи:

- создавать соединение с источником данных;
- создавать объект, реализующий SQL-команду;
- выполнять SQL-команду;
- сохранять результат выполнения SQL-команды в кэше;
- конструировать виртуальное представление кэша, чтобы пользователь мог фильтровать, сортировать данные и перемещаться по ним;
- редактировать данные;
- обновлять источник данных в соответствии с внесенными изменениями;
- фиксировать или отменять изменения, внесенные в процессе транзакции и последующего ее закрытия.

В объектную модель ADO входят перечисленные в табл. 17.1 объекты.

Таблица 17.1. *Объектная модель ADO*

Объект	Описание
`Connection`	Среда, в которой выполняется обмен данными с источником данных. Соединение с источником данных необходимо создать до начала выполнения любых операций
`Command`	Способ управления источником данных. Позволяет добавлять, удалять и считывать данные из источника
`Parameter`	Переменные компоненты объекта `Command`, которые уточняют способ выполнения команд. Перед выполнением каждой из команд эти параметры можно модифицировать
`Recordset`	Локальный кэш для данных, считанных из источника
`Field`	Столбец таблицы данных `Recordset`. Поля содержат свойства, определяющие их параметры, например, тип
`Error`	Инкапсулирует данные о сгенерированной ошибке
`Property`	Определяет объекты `Connection`, `Command`, `Recordset` и `Field`. Каждый объект ADO обладает набором свойств, описывающим его или им управляющих. Встроенные свойства существуют всегда. Динамические свойства добавляются источником данных OLE DB и существуют только до конца работы с ним
`Collection`	Служит для объединения сходных объектов ADO в группы. Имеются четыре подобные группы объектов: `Errors`, `Parameters`, `Fields` и `Properties`. Каждый объект `Recordset` включает в себе семейство `Fields`, представляющее все поля таблицы. Объект же `Command` содержит семейство `Parameters`, возвращающее параметры команды
`Stream`	Поток ввода/вывода текста или данных

Создание ссылки на библиотеку ADO

Прежде чем применять технологию ADO, на компьютере должна быть инсталлирована библиотека ADO. Она инсталлируется вместе с Microsoft Office и может функционировать в любой среде разработки, в которой поддерживается взаимодействие с объектами ActiveX/COM, включая Visual Basic, Visual C++ и Active Server Pages.

Если библиотека ADO инсталлирована, то ее объекты можно использовать после установки ссылки на эту библиотеку. Для этого:

1. Выберите команду **Tools | References**. Появится окно **References**.
2. Установите флажок **Microsoft ActiveX Data Objects 2.0 Library**. Нажмите кнопку **OK**.

После выполнения этих действий ADO может быть использована в вашем коде.

Объект *Connection* и установка подключения к базе данных

Объект `Connection` представляет собой среду, в которой выполняется обмен данными с источником данных. Соединение с источником данных необходимо создать до начала выполнения любых операций. После прекращения обмена данными с источником, соединение должно быть закрыто.

Для установки подключения к источнику данных применяется метод `Open` объекта `Connection`, а для закрытия соединения — метод `Close`.

Свойство `Provider` является текстовой строкой, задающей тип провайдера OLE DB, который будет применен для подключения. При использовании провайдера ODBC его можно не указывать, т. к. он используется по умолчанию и называется `MSDASQL`.

Свойство `ConnectionString` указывает способ подключения к источнику данных.

При использовании Jet-провайдера строка подключения может состоять из полного имени файла. Например, в следующем коде (листинг 17.1) при инициализации формы устанавливается соединение с базой данных **Борей** с использованием Jet-провайдера, а при закрытии формы это соединение также закрывается.

Листинг 17.1. Подключение с использованием Jet-провайдера

```
Private cn As ADODB.Connection

Private Sub UserForm_Initialize()
   Set cn = New ADODB.Connection
   cn.Provider = "Microsoft.Jet.OLEDB.4.0"
   cn.ConnectionString = "c:\Борей.mdb"
   cn.Open
End Sub

Private Sub UserForm_Terminate()
   cn.Close
End Sub
```

Объект *Recordset* и его создание

После установки соединения с источником данных можно создать объект `Recordset`. Этот объект представляет собой локальный кэш для исходной таблицы базы данных или результирующего набора считанных из источника

записей, возвращаемых в результате запроса. Таким образом, объект `Recordset` — это не реальный, а виртуальный набор записей из базы данных, позволяющий управлять данными в базе данных на уровне записи. На уровне полей управление данными осуществляется объектом `Field`, причем свойство `Fields` возвращает семейство всех полей данного набора записей. Средства ADO позволяют создавать объект `Recordset` непосредственно или используя объекты `Connection` и `Command`. Объекты `Recordset` поддерживают четыре типа курсоров, перечисленных в табл. 17.2.

Таблица 17.2. *Типы курсоров объекта* `Recordset`

Тип	Описание
Dynamic (Динамический)	Позволяет видеть изменения в данных, сделанные другими пользователями и поддерживает все типы движений, которые не основываются на закладках. Может разрешать использование закладок, если они поддерживаются поставщиком данных
Keyset (Управляемые ключи)	Является динамическим курсором, но не позволяет видеть записи, добавленные или удаленные другими пользователями. В то же время позволяет видеть изменения в данных, сделанные другими пользователями. Поддерживает закладки и все виды перемещения по записям
Static (Статический)	Обеспечивает фиксированную копию данных все время, пока открыт набор записей. Дополнения, изменения и удаления, выполняемые другими пользователями, не имеют никакого воздействия на набор записей, так как он не обновляется после создания. Поддерживает закладки и все виды перемещения по записям
Forward-only (Однонаправленный)	Статический режим с последовательным доступом. Перемещаться от записи к записи можно только вперед. Это самый скоростной режим

Объект `Recordset` открывается методом `Open`. При этом надо указать таблицу или запрос, на основе которого строится этот объект, либо в параметре метода `Open`, либо при помощи свойства `Source`. Также требуется указать при помощи свойства `ActiveConnection` или одноименного параметра метода `Open`, на основе какого соединения конструируется объект `Recordset`.

Пример использования объекта *Recordset* для просмотра базы данных

Приводимый проект на примере таблицы **Клиенты** базы данных **Борей** (рис. 17.1) демонстрирует, как можно создать объект `Recordset`. Итак, сконструируйте форму, на которой расположите список, а в модуле формы наберите

следующий код (листинг 17.2). При заполнении списка надо последовательно перемещаться по записям объекта `Recordset`, что в коде производится методом `MoveNext`. Свойство `EOF` проверяет, находится ли указатель непосредственно после последней записи объекта `Recordset`.

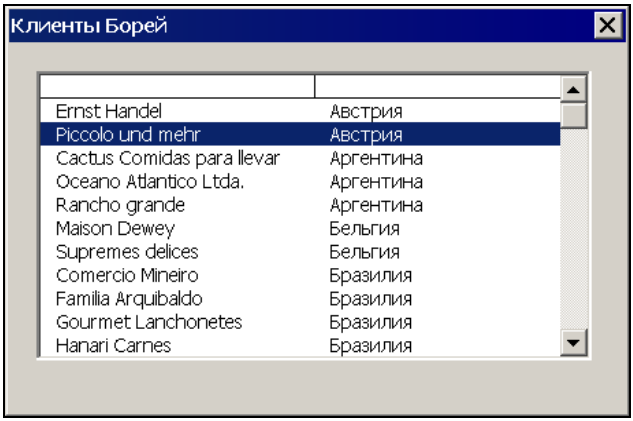


Рис. 17.1. Поля **Название** и **Страна** таблицы **Клиенты** базы данных **Борей**

Листинг 17.2. Создание объекта *Recordset*

```
Private cn As ADODB.Connection

Private Sub UserForm_Initialize()
   Me.Caption = "Клиенты Борей"
   ListBox1.ColumnCount = 2

   Set cn = New ADODB.Connection
   cn.Provider = "Microsoft.Jet.OLEDB.4.0"
   cn.ConnectionString = "c:\Борей.mdb"
   cn.Open

   Dim rs As New ADODB.Recordset
   rs.Source = "SELECT Название, Страна FROM Клиенты " & _
               "Order By Страна, Название"
   Set rs.ActiveConnection = cn
   rs.Open
   ListBox1.Clear
   ListBox1.ColumnHeads = True
```

```
    Dim i As Long
    i = 0
    Do Until rs.EOF
       ListBox1.AddItem rs.Fields("Название").Value
       ListBox1.List(i, 1) = rs.Fields("Страна").Value
       rs.MoveNext
       i = i + 1
    Loop
    rs.Close
End Sub

Private Sub UserForm_Terminate()
    cn.Close
    Set cn = Nothing
End Sub
```

Методы, свойства и события объекта *Recordset*

Объект `Recordset` обладает обширной коллекцией методов, свойств и событий (табл. 17.3, 17.4 и 17.5), позволяющих редактировать, создавать, обновлять записи, а также осуществлять полный контроль над этими и другими операциями.

Таблица 17.3. *Методы объекта* `Recordset`

Метод	Описание
`AddNew`	Подготавливает новую запись для добавления в набор записей. После внесения изменений следует вызвать метод `Update` для сохранения изменений и добавления записи в объект `Recordset`. До вызова метода `Update` изменения в базу данных не заносятся
`Cancel`	Отмена асинхронного метода `Open`
`CancelBatch`	Отмена всех не принятых изменений в наборе записей
`CancelUpdate`	Отмена результатов операций `AddNew` и `Edit` без сохранения изменений
`Clone`	Создает копию набора записей
`Close`	Закрывает набор записей
`CompareBookmarks`	Определяет, какая из двух закладок указывает на предыдущую строку набора записей
`Delete`	Удаляет текущую запись

Таблица 17.3 *(окончание)*

Метод	Описание
`Find`	Находит набор записей, удовлетворяющий специфицированному критерию
`GetRows`	Заполняет массив записями из набора записей
`GetString`	Возвращает набор записей в виде строк с разделителями
`Move`	Перемещение от текущей записи к начальной. Можно указать число записей для перемещения
`MoveFirst`	Переход к первой записи
`MoveLast`	Переход к последней записи
`MoveNext`	Переход к следующей записи
`MovePrevious`	Переход к предыдущей записи
`NextRecordset`	Возвращение следующего набора записей из объекта `Connection`
`Open`	Открытие набора записей
`Requery`	Перезапуск исходного запроса, создавшего данный набор записей
`Resync`	Синхронизация набора записей с соответствующими ему данными
`Save`	Сохранение набора записей в файл
`Seek`	Поиск записи с использованием индекса
`Supports`	Определяет, какие операции поддерживает набор записей
`Update`	Подтверждает работу, выполненную методами `AddNew` и `Edit`
`UpdateBatch`	Подтверждает все текущие изменения в групповом курсоре

Таблица 17.4. *Свойства объекта* `Recordset`

Свойство	Описание
`AbsolutePage`	Количество страниц данных в текущей записи из набора записей
`AbsolutePosition`	Порядковый номер текущей записи в наборе записей
`ActiveCommand`	Команда, создавшая данный набор записей
`ActiveConnection`	Подключение, при помощи которого был создан данный набор записей

Таблица 17.4 *(продолжение)*

Свойство	Описание
BOF	Возвращает значение True, если указатель текущей записи расположен перед первой записью набора записей, и значение False, если указатель текущей записи расположен на первой записи набора или на любой записи после нее
Bookmark	Закладка, уникальный идентификатор текущей записи набора записей
CacheSize	Число записей, кэшированных в оперативной памяти
CursorLocation	Константа, указывающая на местоположение курсора. Допустимыми значениями являются следующие константы: adUseClient и adUseServer
CursorType	Тип курсора. Допустимыми значениями являются adOpenDynamic, adOpenForwardOnly, adOpenKeyset и adOpenStatic
DataMember	Указывает принадлежность к компоненту набора записей при использовании связанных наборов данных
DataSource	Специфицирует источник данных, используемый для связанного набора данных
EditMode	Определяет, была ли отредактирована текущая запись. Допустимыми значениями являются следующие константы: adEditAdd, adEditDelete, adEditInProgress и adEditNone
EOF	Возвращает значение True, если указатель текущей записи расположен после последней записи набора, и значение False, если указатель текущей записи расположен на последней записи набора или на любой записи перед ней
Fields	Семейство полей
Filter	Выбор по критерию поднабора записей из данного набора
Index	Определяет индекс, используемый при выполнении метода Seek
LockType	Задает тип блокировки, используемый при редактировании записей. Допустимыми значениями являются следующие константы: adLockBatchOptimistic, adLockOptimistic, adLockPessimistic и adLockReadOnly
MarshalOptions	Управляет маршализацией данных на сервере для набора записей на клиентской стороне
MaxRecords	Задает максимальное число записей, которые можно возвратить в набор записей
PageCount	Число страниц в наборе записей

Таблица 17.4 *(окончание)*

Свойство	Описание
PageSize	Размер страницы набора записей
Properties	Возвращает семейство Properties, описывающее набор записей
RecordCount	Число записей в наборе. Если ADO не может его определить, то возвращается значение –1
Sort	Задает поле для сортировки набора записей
Source	Команда или запрос SQL, создавший данный набор записей
Status	Результат операции группового обновления данных
NoMatch	Возвращаемое значение True, если нужная запись не найдена, и False в противном случае
StayInSync	Применяется при обновлении иерархического набора

Таблица 17.5. *События объекта* Recordset

Событие	Описание
EndOfRecordset	Происходит при сбое метода MoveNext по причине отсутствия последующей записи
FetchComplete	Генерируется при достижении записей при помощи асинхронной операции
FetchProgress	Генерируется при выполнении асинхронной операции
FieldChangeComplete	Происходит после изменения значения поля
MoveComplete	Генерируется после перемещения указателя записи
RecordChangeComplete	Происходит после изменения целой записи
RecordsetChangeComplete	Генерируется после изменения во всем наборе записей
WillChangeField	Происходит перед изменением значения поля
WillChangeRecord	Происходит перед изменением целой записи
WillChangeRecordset	Происходит перед изменением во всем наборе записей
WillMove	Происходит перед перемещением указателя записи

Последовательный просмотр записей базы данных

Перемещаться по элементам набора записей можно при помощи методов `MoveFirst`, `MoveLast`, `MoveNext` и `MovePrevious` объекта `Recordset`. Продемонстрируем способ навигации по записям базы данных на примере таблицы **Клиенты** базы данных **Борей**. Создайте форму, на которой расположите две надписи, два поля ввода и четыре кнопки (рис. 17.2). Кнопки реализуют следующие функции:

- кнопка << — переход к первой записи;
- кнопка < — переход к предыдущей записи;
- кнопка > — переход к следующей записи;
- кнопка >> — переход к последней записи.

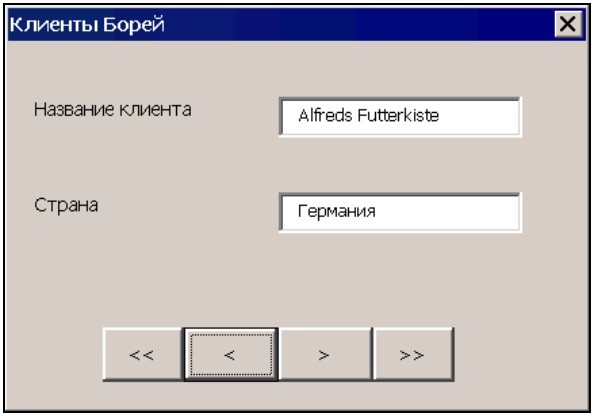


Рис. 17.2. Последовательный просмотр записей базы данных

При помощи окна **Properties** установите значения свойств элементов управления, как показано в табл. 17.6.

Таблица 17.6. *Значения свойств, установленные в окне* ***Properties***

Элемент управления	Свойство	Значение
Надпись	`Caption`	`Название`
Поле ввода	`Name`	`txtClientName`
Надпись	`Caption`	`Страна`
Поле ввода	`Name`	`txtCountry`
Кнопка	`Name`	`cmdFirst`
	`Caption`	`<<`

Таблица 17.6 *(окончание)*

Элемент управления	Свойство	Значение
Кнопка	Name	CmdPrevious
	Caption	<
Кнопка	Name	cmdNext
	Caption	>
Кнопка	Name	cmdLast
	Caption	>>

Установите ссылку на библиотеку доступа к данным. С этой целью в редакторе Visual Basic выберите команду **Tools | References** и убедитесь, что в списке **Available References** установлен флажок **Microsoft ActiveX Data Objects 2.0 Library**. Для завершения проекта осталось только в модуле формы набрать следующий код (листинг 17.3).

Листинг 17.3. Последовательный просмотр записей базы данных

```
Private cn As ADODB.Connection
Private rs As New ADODB.Recordset


Private Sub UserForm_Initialize()
   Set cn = New ADODB.Connection
   cn.Provider = "Microsoft.Jet.OLEDB.4.0"
   cn.ConnectionString = "D:\Борей.mdb"
   cn.Open
   Me.Caption = "Клиенты Борей"
   rs.CursorType = adOpenStatic
   rs.Source = "SELECT Название, Страна FROM Клиенты Order By Название"
   Set rs.ActiveConnection = cn
   rs.Open
   cmdFirst_Click
End Sub

Private Sub cmdFirst_Click()
   rs.MoveFirst
   ShowRecord
```

```
End Sub

Private Sub cmdPrevious_Click()
   If Not rs.BOF Then
      rs.MovePrevious
      If Not rs.BOF Then
        ShowRecord
      Else
         rs.MoveFirst
      End If
   End If
End Sub

Private Sub cmdNext_Click()
   If Not rs.EOF Then
      rs.MoveNext
      If Not rs.EOF Then
         ShowRecord
      Else
         rs.MoveLast
      End If
   End If
End Sub

Private Sub cmdLast_Click()
   rs.MoveLast
   ShowRecord
End Sub


Private Sub ShowRecord()
   txtClientName = rs.Fields("Название").Value
   txtCountry.Text = rs.Fields("Страна").Value
End Sub

Private Sub UserForm_Terminate()
   rs.Close
   cn.Close
   Set rs = Nothing
   Set cn = Nothing
End Sub
```

Доступ к данным, расположенным на рабочих листах закрытой книги

Механизм ADO можно использовать не только при доступе к данным из баз данных, но и рабочих книг, в частности из закрытых рабочих книг. Следующий код демонстрирует подобный подход. Процедура `GetDataFromWorkbook` имеет четыре параметра. Параметр `SourceFile` задает ссылку на рабочую книгу, параметр `SourceRange` — ссылку на диапазон с импортируемыми данными. Эта ссылка может задаваться либо по имени диапазона (в этом случае диапазон может располагаться в любом месте рабочей книги), либо по его обозначению в формате A1 (в этом случае диапазон может располагаться только на первом листе рабочей книги). Параметр `TargetRange` задает ссылку на ячейку, расположенную в верхнем левом углу того диапазона, куда будут выводиться импортируемые данные. Параметр `IncludeFieldNames` определяет, надо ли в импортируемом диапазоне учитывать заголовок таблицы.

Процедура `DemoGetDataFromWorkbook` из листинга 17.4 демонстрирует вызов процедуры `GetDataFromWorkbook`. Предполагается, что имеется книга c:\test.xls, у которой в диапазоне **A1:C2** первого рабочего листа содержится одна таблица данных, а в некотором другом диапазона с именем `MyRange` другого рабочего листа содержится другая таблица. Эти две таблицы и импортируются в данную рабочую книгу.

Листинг 17.4. Доступ к данным, расположенным на рабочих листах закрытой книги

```
Sub DemoGetDataFromWorkbook()
   GetDataFromWorkbook "c:\test.xls", "A1:C2", Range("A2"), True
   GetDataFromWorkbook "c:\test.xls", "MyRange", Range("A5"), True
End Sub

Sub GetDataFromWorkbook(SourceFile As String, _
                        SourceRange As String, _
                        TargetRange As Range, _
                        IncludeFieldNames As Boolean)
   Dim con As ADODB.Connection
   Dim rs As ADODB.Recordset
   Dim conString As String
   Dim TargetCell As Range, i As Integer
   conString = "DRIVER={Microsoft Excel Driver (*.xls)};" & _
               "ReadOnly=1;DBQ=" & SourceFile
```

```
   Set con = New ADODB.Connection
   On Error GoTo ErrorHandler
   con.Open conString
   Set rs = con.Execute("[" & SourceRange & "]")
   Set TargetCell = TargetRange.Cells(1, 1)
   If IncludeFieldNames Then
      For i = 0 To rs.Fields.Count - 1
         TargetCell.Offset(0, i).Value = rs.Fields(i).Name
      Next
      Set TargetCell = TargetCell.Offset(1, 0)
   End If
   TargetCell.CopyFromRecordset rs
   rs.Close
   con.Close
   Set TargetCell = Nothing
   Set rs = Nothing
   Set dbConnection = Nothing
   On Error GoTo 0
   Exit Sub
ErrorHandler:
   MsgBox "Данные не доступны!"
End Sub
```

Работа с курсорами

ADO предусматривает поддержку нескольких типов курсоров. Помимо обеспечения последовательного перемещения по набору записей курсоры позволяют контролировать способ управления набором записей.

Расположение курсора задается свойством `CursorLocation`, у которого имеются следующие два допустимых значения:

- `adUseServer` — курсор создается на стороне сервера;
- `adUseClient` — курсор создается на стороне клиента. Операциями с курсором управляют ADO и OLE DB. По сравнению с серверным курсором допустимо, например, создание отключенного набора записей, позволяющего манипулировать записями без постоянного подключения к серверу.

Свойство `CursorType` устанавливает тип курсора. Допустимыми значениями являются следующие константы:

- `adOpenDynamic` — динамический тип. Отслеживаются все изменения записей, произведенные другими пользователями;

- `adOpenForwardOnly` — однонаправленный тип. Перемещаться можно только вперед;
- `adOpenKeyset` — ключевой тип. Нельзя просматривать записи, добавленные в набор другими пользователями, но можно отслеживать сделанные ими обновления и удаления;
- `adOpenStatic` — статический. Работает с копией набора данных.

Объект *Field*

Объект `Field` представляет собой поле набора записей. Все поля образуют семейство `Fields`. У этого объекта имеется всего два метода и свойства, которые перечислены в табл. 17.7 и 17.8.

Таблица 17.7. *Методы объекта* `Field`

Метод	Описание
`AppendChunk`	Сохраняет порцию данных в поле двоичного числа типа `Long`
`GetChunk`	Изменяет порцию данных из поля двоичного числа типа `Long`

Таблица 17.8. *Свойства объекта* `Field`

Свойство	Описание
`ActualSize`	Размер данных, сохраненных в поле
`Attributes`	Список атрибутов поля
`DataFormat`	Управляет форматированием данных
`DefinedSize`	Максимальные размеры поля
`Name`	Имя поля
`NumericScale`	Разрядность числового поля
`OriginalValue`	Значение в поле перед тем, как его изменил пользователь
`Precision`	Число значащих цифр числового поля
`Properties`	Семейство `Properties`, описывающее свойства поля
`Type`	Тип данных поля
`UnderlyingValue`	Текущее значение, хранимое в базе данных для данного поля
`Value`	Текущее значение поля набора записей

Создание браузера базы данных

Для таблицы **Клиенты** базы данных **Борей** создадим простейший браузер. В этом браузере в список будут выводиться названия клиентов и их коды, причем последние нужны только для идентификации выбранной записи и поэтому выводятся в скрытый столбец списка. При выборе элемента из списка в полях ввода отображаются город и страна клиента, которые как раз и идентифицируются по скрытому в списке коду клиента (рис. 17.3).

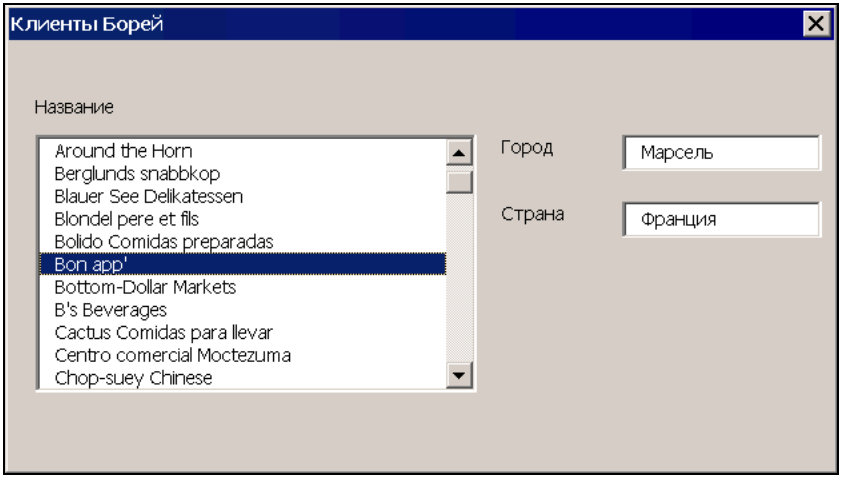


Рис. 17.3. Создание браузера базы данных

Итак, создайте форму, на которой расположите список, три надписи и два поля ввода и при помощи окна **Properties** установите им значения свойств, как показано в табл. 17.9.

Таблица 17.9. *Значения свойств, установленные в окне* ***Properties***

Элемент управления	Свойство	Значение
Надпись	`Caption`	`Название`
Список	`Name`	`lstClient`
Надпись	`Caption`	`Город`
Поле ввода	`Name`	`txtTown`
Надпись	`Caption`	`Страна`
Поле ввода	`Name`	`txtCountry`

Установите ссылку на библиотеку доступа к данным. С этой целью в редакторе Visual Basic выберите команду **Tools | References** и убедитесь, что в спи-

ске **Available References** установлен флажок **Microsoft ActiveX Data Objects 2.0 Library**. Для завершения проекта осталось только в модуле формы набрать следующий код (листинг 17.5).

Листинг 17.5. Браузер базы данных

```
Private cn As ADODB.Connection
Private rs As New ADODB.Recordset

Private Sub UserForm_Initialize()
   Set cn = New ADODB.Connection
   cn.Provider = "Microsoft.Jet.OLEDB.4.0"
   cn.ConnectionString = "D:\Борей.mdb"
   cn.Open
   Me.Caption = "Клиенты Борей"
   Dim idx As Long
   lstClient.ColumnCount = 2
   lstClient.ColumnWidths = "80;0"
   rs.Source = _
      "SELECT КодКлиента, Название FROM Клиенты Order By Название"
   Set rs.ActiveConnection = cn
   rs.Open
   idx = 0
   Do Until rs.EOF
      lstClient.AddItem rs.Fields("Название")
      lstClient.List(idx, 1) = rs.Fields("КодКлиента")
      idx = idx + 1
      rs.MoveNext
   Loop
   rs.Close
   lstClient.ListIndex = 0
End Sub

Private Sub lstClient_Click()
   Dim id As String
   id = lstClient.List(lstClient.ListIndex, 1)
   rs.Source = "SELECT Город, Страна FROM Клиенты WHERE КодКлиента=" _
               & Chr(39) & id & Chr(39)
   rs.Open
```

```
   txtTown.Text = rs.Fields("Город").Value
   txtCountry.Text = rs.Fields("Страна").Value
   rs.Close
End Sub

Private Sub UserForm_Terminate()
   cn.Close
   Set cn = Nothing
   Set rs = Nothing
End Sub
```

Браузер просмотра нескольких таблиц базы данных

В *разд.* "*Последовательный просмотр записей базы данных*" приводился пример просмотра одной таблицы базы данных. Настало время продемонстрировать технику работы с базой данных, состоящей из нескольких таблиц. Сделаем это на примере учебной базы данных **Борей**. Создайте форму, на которой расположите два списка и две надписи (рис. 17.4).

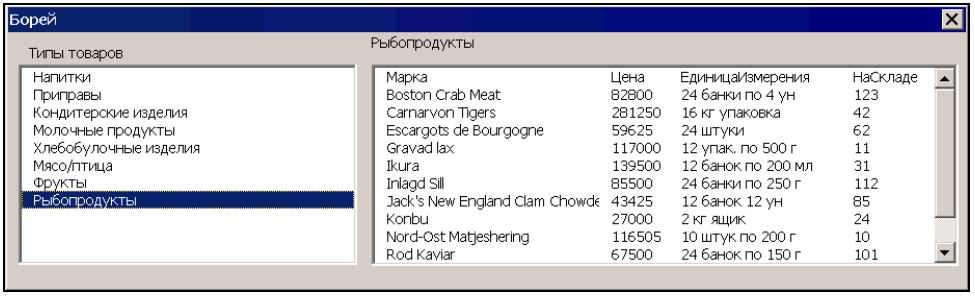


Рис. 17.4. Браузер базы данных **Борей**

В левый список при инициализации формы выведем все записи из поля **Категория** таблицы **Типы**. Кроме того, в скрытый столбец списка выведем соответствующие значения из поля **КодТипа**, которое является первичным ключом этой таблицы.

При выборе категории товара из левого списка в надпись, расположенную над правым списком, выводится указанная категория товара. Создается запрос к таблице **Товары**. По этому запросу выбираются все записи из полей **Марка**, **Цена**, **ЕдиницаИзмерения**, **НаСкладе** такие, что значение поля **КодТипа** (которое является внешним ключом) совпадает с указанным значением поля **КодТипа** из таблицы **Типы**. Результат запроса выводится во второй список.

Для реализации проекта при помощи окна **Properties** установите значения свойств элементов управления, как показано в табл. 17.10.

Таблица 17.10. *Значения свойств, установленных в окне* ***Properties***

Элемент управления	Свойство	Значение
Надпись	Name	lblCategory
	Caption	Типы товаров
Список	Name	lstCategory
Надпись	Name	lblGood
Список	Name	lstGood

Установите ссылку на библиотеку доступа к данным. С этой целью в редакторе Visual Basic выберите команду **Tools | References** и убедитесь, что в списке **Available References** установлен флажок **Microsoft ActiveX Data Objects 2.0 Library**. Для завершения проекта осталось только в модуле формы набрать следующий код (листинг 17.6).

Листинг 17.6. Еще один браузер базы данных

```
Private cn As ADODB.Connection
Private rs As New ADODB.Recordset

Private Sub UserForm_Initialize()
   Set cn = New ADODB.Connection
   cn.Provider = "Microsoft.Jet.OLEDB.4.0"
   cn.ConnectionString = "D:\Борей.mdb"
   cn.Open

   With lstCategory
      .ColumnCount = 2
      .ColumnWidths = "120;0"
   End With
   With lstGood
      .ColumnCount = 4
      .ColumnWidths = "130;40;100;40"
   End With

   Dim sqlQuery As String
   Dim i As Long
```

```
   Me.Caption = "Борей"
   rs.CursorType = adOpenKeyset
   rs.LockType = adLockOptimistic
   sqlQuery = "SELECT КодТипа, Категория FROM Типы"
   rs.Source = sqlQuery
   Set rs.ActiveConnection = cn
   rs.Open
   Application.Cursor = xlWait
   i = 0
   Do Until rs.EOF
      lstCategory.AddItem rs.Fields("Категория")
      lstCategory.List(i, 1) = rs.Fields("КодТипа")
      i = i + 1
      rs.MoveNext
   Loop
   rs.Close
   Application.Cursor = xlDefault
   lstCategory.ListIndex = 0
End Sub

Private Sub lstCategory_Click()
   Dim sqlQuery As String
   Dim idCategory As Long
   If lstCategory.ListIndex = -1 Then Exit Sub
   lblGood.Caption = lstCategory.Text
   idCategory = lstCategory.List(lstCategory.ListIndex, 1)
   sqlQuery = "SELECT Марка, Цена, ЕдиницаИзмерения, НаСкладе " _
              & "FROM Товары WHERE КодТипа = " _
              & CStr(idCategory) & " ORDER BY Марка"
   Application.Cursor = xlWait
   rs.Source = sqlQuery
   Set rs.ActiveConnection = cn
   rs.Open
   With lstGood
      .Clear
      .AddItem rs.Fields("Марка").Name
      .List(0, 1) = rs.Fields("Цена").Name
      .List(0, 2) = rs.Fields("ЕдиницаИзмерения").Name
      .List(0, 3) = rs.Fields("НаСкладе").Name
```

```
    End With

    Dim i As Long
    i = 1
    Do Until rs.EOF
       With lstGood
          .AddItem rs.Fields("Марка").Value
          .List(i, 1) = rs.Fields("Цена").Value
          .List(i, 2) = rs.Fields("ЕдиницаИзмерения").Value
          .List(i, 3) = rs.Fields("НаСкладе").Value
       End With
       i = i + 1
       rs.MoveNext
    Loop
    rs.Close
    Application.Cursor = xlDefault
End Sub
```

Редактирование, создание, обновление и удаление записей

При создании и открытии набора записей значения нескольких свойств, определяющих этот набор, устанавливаются по умолчанию. Если же требуется внести в набор какие-либо изменения, то значения этих свойств, а именно `LockType` и `CursorType`, также надо изменить. Свойство `LockType` может принимать следующие значения:

- `adLockBatchOptimistic` — разрешено пакетное обновление;
- `adLockOptimistic` — устанавливает блокировку отдельных записей, причем запись блокируется при вызове метода `Update`;
- `adLockPessimistic` — устанавливает блокировку отдельных записей, причем запись блокируется сразу же после начала редактирования;
- `adLockReadOnly` — объект `Recordset` доступен только для чтения. Редактирование запрещено. Это значение устанавливается по умолчанию.

По умолчанию набор записей открывается со значением свойства `LockType`, равным `adLockReadOnly`. Чтобы обойти это ограничение, надо установить его значение равным либо `adLockOptimistic`, либо `adLockPessimistic`. Различие между этими двумя значениями проявляется только при работе в многопользовательском режиме и состоит в том, что записи блокируются либо при обновлении, либо при редактировании.

Обновление записи производится методом `Update`, создание новой записи — методом `AddNew`, а удаление текущей записи — методом `Delete`.

Продемонстрируем работу этих методов на примере табличной базы данных **Phone**, состоящей из единственной таблицы **Телефон** с тремя полями: **Фамилия**, **Имя** и **Телефон**.

Итак, создайте форму, на которой расположите семь кнопок, три надписи и три поля ввода (рис. 17.5). Установите им при помощи окна **Properties** значения свойств, как показано в табл. 17.11.

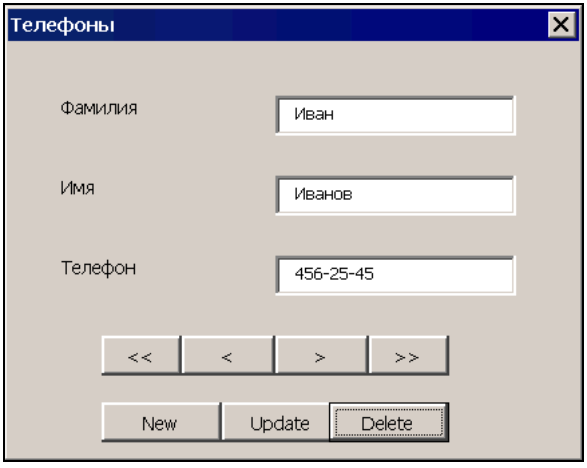


Рис. 17.5. Редактирование, создание, обновление и удаление записей

Таблица 17.11. *Значения свойств, установленные в окне* ***Properties***

Элемент управления	Свойство	Значение
Надпись	`Caption`	`Название`
Поле ввода	`Name`	`txtClientName`
Надпись	`Caption`	`Страна`
Поле ввода	`Name`	`txtCountry`
Кнопка	`Name`	`cmdFirst`
	`Caption`	`<<`
Кнопка	`Name`	`cmdPrevious`
	`Caption`	`<`
Кнопка	`Name`	`cmdNext`
	`Caption`	`>`

Таблица 17.11 *(окончание)*

Элемент управления	Свойство	Значение
Кнопка	Name	CmdLast
	Caption	>>
Кнопка	Name	cmdNew
	Caption	New
Кнопка	Name	cmdUpdate
	Caption	Update
Кнопка	Name	cmdDelete
	Caption	Delete

Кнопки <<, <, > и >> будут осуществлять навигацию по таблице базы данных. Остальные кнопки выполняют следующие функции:

- кнопка **Update** предназначена для обновления записи в базе данных;
- кнопка **Delete** предназначена для удаления записи из базы данных;
- кнопка **New** предназначена для создания новой записи. При этом при нажатии кнопки **New** все кнопки окна, за исключением кнопки **Update**, блокируются для пользователя. Таким образом, пользователь не может осуществлять никаких операций над записями до тех пор, пока не запишет новую запись в базу данных. При нажатии же кнопки **Update** происходит разблокирование всех кнопок.

В приводимом ниже коде (см. листинг 17.7) за процесс управления блокировкой кнопок отвечает процедура `SetEnabled`, а переменная `IsDisable` определяет достижимость кнопок для пользователя.

Не забудьте установить ссылку на библиотеку доступа к данным. С этой целью в редакторе Visual Basic выберите команду **Tools | References** и убедитесь, что в списке **Available References** установлен флажок **Microsoft ActiveX Data Objects 2.0 Library**.

Обратите внимание, что использование свойства `Tag`, а также префиксов в именах элементов управления, позволяет идентифицировать требуемый элемент управления (кнопку) при переборе в цикле

```
For Each ctrl In Controls
Next
```

Идентификация же кнопки **Update** в этом цикле производится по значению свойства `Tag`.

Листинг 17.7. Редактирование, создание и удаление записей

```
Private cn As ADODB.Connection
Private rs As New ADODB.Recordset
Private IsDisable As Boolean

Private Sub UserForm_Initialize()
   Set cn = New ADODB.Connection
   cn.Provider = "Microsoft.Jet.OLEDB.4.0"
   cn.ConnectionString = "D:\phone.mdb"
   cn.Open
   Me.Caption = "Телефоны"
   rs.CursorType = adOpenKeyset
   rs.LockType = adLockOptimistic
   rs.Source = "SELECT * FROM Телефон ORDER BY Фамилия"
   Set rs.ActiveConnection = cn
   rs.Open
   cmdUpdate.Tag = "Update"
   cmdFirst_Click
   IsDisable = False
End Sub

Private Sub cmdFirst_Click()
   rs.MoveFirst
   ShowRecord
End Sub

Private Sub cmdPrevious_Click()
   If Not rs.BOF Then
      rs.MovePrevious
      If Not rs.BOF Then
        ShowRecord
      Else
         rs.MoveFirst
      End If
   End If
End Sub

Private Sub cmdNext_Click()
```

```
   If Not rs.EOF Then
      rs.MoveNext
      If Not rs.EOF Then
         ShowRecord
      Else
         rs.MoveLast
      End If
   End If
End Sub

Private Sub cmdLast_Click()
   rs.MoveLast
   ShowRecord
End Sub



Private Sub cmdNew_Click()
   ShowEmptyRecord
   rs.AddNew
   FillRecord
   txtLName.SetFocus
   IsDisable = True
   SetEnabled True, False
End Sub

Private Sub cmdUpdate_Click()
   If Not IsDisable Then rs.Update
   FillRecord
   SetEnabled True, True
   If IsDisable Then
      rs.MoveLast
      IsDisable = False
   End If
End Sub

Private Sub cmdDelete_Click()
   If rs.RecordCount >= 1 Then
      If MsgBox("Удалить текущую запись?", vbYesNo + vbQuestion) _
          = vbYes Then
```

```
         rs.Delete
         If rs.RecordCount > 0 Then
            cmdNext_Click
         Else
            ShowEmptyRecord
         End If
      End If
   End If
End Sub

Private Sub SetEnabled(IsUpdateOn As Boolean, IsOthersOn As Boolean)
    Dim ctrl As Control
    For Each ctrl In Controls
       If LCase(Left(ctrl.Name, 3)) = "cmd" Then
          If ctrl.Tag = "Update" Then
                ctrl.Enabled = IsUpdateOn
          Else
                ctrl.Enabled = IsOthersOn
          End If
       End If
    Next ctrl
End Sub

Private Sub ShowRecord()
   txtFName.Text = rs.Fields("Фамилия").Value
   txtLName.Text = rs.Fields("Имя").Value
   txtPhone.Text = rs.Fields("Телефон").Value
End Sub

Private Sub FillRecord()
   rs.Fields("Фамилия").Value = txtFName.Text
   rs.Fields("Имя").Value = txtLName.Text
   rs.Fields("Телефон").Value = txtPhone.Text
End Sub

Private Sub ShowEmptyRecord()
   txtLName.Text = Empty
   txtFName.Text = Empty
   txtPhone.Text = Empty
```

```
End Sub

Private Sub UserForm_Terminate()
   cm = Close
   rs.Close
   Set rs = Nothing
   Set cn = Nothing
End Sub
```

Вывод набора записей на рабочий лист

Для вывода набора записей на рабочий лист можно воспользоваться методом `CopyFromRecordset` объекта `Range`.

`CopyFromRecordset(Data, MaxRows, MaxColumns)`, где:

- *`Data`* — обязательный параметр, задающий копируемый набор записей;
- *`MaxRows`* — необязательный параметр, определяющий максимальное число копируемых записей;
- *`MaxColumns`* — необязательный параметр, определяющий максимальное число копируемых полей.

В следующем примере (листинг 17.8) на рабочий лист книги выводятся записи из таблицы **Клиенты** базы данных **Борей**.

Листинг 17.8. Вывод набора записей на рабочий лист

```
Private Sub LoadRecordsetToWorkbook()
   Dim cn As New ADODB.Connection
   cn.Provider = "Microsoft.Jet.OLEDB.4.0"
   cn.ConnectionString = "c:\Борей.mdb"
   cn.Open

   Dim rs As New ADODB.Recordset
   rs.Source = "SELECT Название, Страна FROM Клиенты"
   Set rs.ActiveConnection = cn
   rs.Open
   Dim NewBook As Workbook
   Dim i As Integer
   Set NewBook = Workbooks.Add

   For i = 0 To rs.Fields.Count - 1
```

```
      NewBook.Sheets(1).Range("A1").Offset(0, i).Value _
         = rs.Fields(i).Name
   Next
   NewBook.Sheets(1).Range("A2").CopyFromRecordset rs
   rs.Close
   Set rs = Nothing
   cn.Close
   Set cn = Nothing
End Sub
```

Использование объекта *Command*

Объект `Command` позволяет применять SQL команды. Основным свойством этого объекта является свойство `CommandText`, которое задает SQL команду, пересылаемую в источник данных через поставщика OLE DB. В следующем коде (листинг 17.9), например, в базу данных **phonebook** вводится очередная запись, причем в ней только специфицируются значения полей **Фамилия** и **Телефон**.

Листинг 17.9. Использование объекта *Command*

```
Sub InsertRecord()
   Set cn = New ADODB.Connection
   cn.Provider = "Microsoft.Jet.OLEDB.4.0"
   cn.ConnectionString = "C:\phonebook.mdb"
   cn.Open
   Dim cmd  As New ADODB.Command
   cmd.CommandText = "INSERT INTO tblPhone(Фамилия, Телефон) " & _
                     "VALUES('Иванова','121-11-11')"
   Set cmd.ActiveConnection = cn
   cmd.Execute
   Set cnd = Nothing
   cn.Close
   Set cn = Nothing
End Sub
```

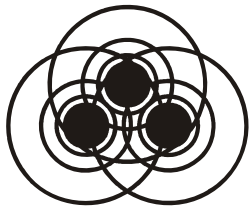
Вставка данных в рабочий лист без ADO

Для вставки данных из базы данных в рабочий лист можно обойтись и без ADO, а воспользоваться объектом `QueryTable`, который инкапсулирует информацию о запросах к различным источникам данных, например как показано в коде из листинга 17.10.

Листинг 17.10. Вставка данных в рабочий лист

```
Sub ImportData()
   Dim strSQL As String, strCNN As String
   Dim qt As QueryTable
   strSQL = "Select Название, Город FROM Клиенты"
   strCNN = _
      "OLEDB;Provider=Microsoft.Jet.OLEDB.4.0;Data Source=D:\БОРЕЙ.MDB;"
   Set qt = ActiveSheet.QueryTables.Add( _
             Connection:=strCNN, _
             Destination:=Range("B1"), _
             Sql:=strSQL)
   qt.Refresh
   Set qt = Nothing
End Sub
```

Глава 18



VBA и Access

MS Access работает с двумя форматами данных: с MDB, который использует механизм Microsoft Jet и ADB, использующим механизм Microsoft SQL Server. Под механизмом базы данных можно понимать интерпретатор, который переводит общие команды (например, "Сортировать таблицу по первому полю в алфавитном порядке") в специальные команды, требуемые конкретным форматом базы данных. Этим объясняется различие между макросами в Access и других офисных приложениях. В этой главе демонстрируется, как в Access производится работа с формами и элементами управления, объясняется, что такое связанные и несвязанные формы, отчеты и элементы управления. Также показывается, как можно создавать средства навигации по базам данных.

Макросы в Access

В MS Access макросы существенно отличаются от макросов других приложений MS Office типа MS Excel или MS Word. Если последние — это запись

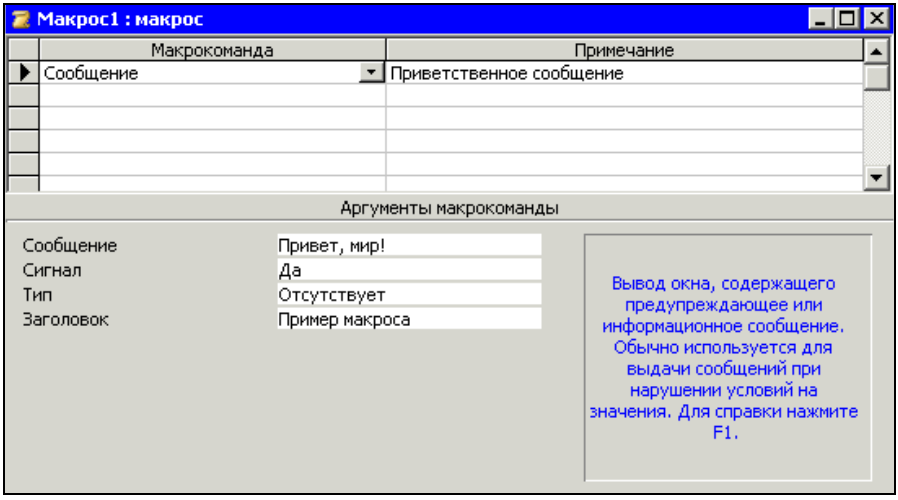


Рис. 18.1. Макросы в MS Access

последовательности действий пользователя, то в MS Access — это набор специальных команд с установленными аргументами. Макросы были очень популярны в ранних версиях MS Access, но, начиная с версии MS Access 2002, их значение постепенно сводится к нулю, и они обычно применяются при программировании быстрых клавиш типа <Ctrl>+<P> — печать текущего документа. На рис. 18.1 показан пример макроса-сообщения, созданного в MS Access.

Применение объекта *DoCmd*

При написании кода многие макрокоманды могут быть вызваны с помощью объекта `DoCmd`. В этом случае макрокоманды выглядят как обычные методы. Например, следующее применение метода `OpenForm` эквивалентно использованию макрокоманды `ОткрытьФорму`.

```
DoCmd.OpenForm FormName:="frmSplash", _
               View:=acNormal, _
               DataMode:=acFormEdit, _
               WindowMode:=acWindowNormal
```

Метод `Beep` позволяет генерировать простейший звуковой сигнал, а метод `Hourglass` — установить указатель мыши в виде песочных часов.

```
DoCmd.Beep
DoCmd.Hourglass True
```

Объектная модель *MS Access*

В вершине объектной модели MS Access находится объект `Application`. На следующем уровне иерархии отметим следующие три основные объекта, с которыми главным образом и приходится работать:

- объект `Form` используется для ввода и редактирования данных;
- объект `Report` используется для отображения отформатированных данных, которые не могут редактироваться;
- объект `DataAccessPage` соединяет в себе средства форм и отчетов для ввода, редактирования и отображения данных. Он разработан для использования в Web-браузере.

Открытие и закрытие *MS Access* из других приложений

При автоматизации MS Access из других приложений MS Office обычно требуется открытие и, возможно, закрытие приложения MS Access или базы данных, с которой ведется работа. Для открытия существующей базы данных формата MDB используется метод `OpenCurrentDatabase`. Следующий

код (листинг 18.1) открывает из MS Excel базу данных **Борей** и выводит из нее отчет **Счет** после чего MS Access закрывается.

Листинг 18.1. Открытие MS Access с выводом отчета

```
Sub OpenReport()
   Dim objAccess As Access.Application
   Set objAccess = New Access.Application
   Const PATH As String = "C:\"
   With objAccess
      .OpenCurrentDatabase PATH & "Борей.mdb"
      .DoCmd.OpenReport "Счет"
   End With
   objAccess.Quit
End Sub
```

В следующем коде (листинг 18.2) производится открытие MS Access с отображением формы **Сотрудники** в которой выводятся все сотрудники, имеющие должность "Представитель".

Листинг 18.2. Открытие MS Access с отображением формы

```
Sub OpenForm()
   Dim objAccess As Access.Application
   Set objAccess = New Access.Application
   Const PATH As String = "C:\"
   objAccess.OpenCurrentDatabase PATH & "Борей.mdb"
   objAccess.DoCmd.OpenForm FormName:="Сотрудники", _
            WhereCondition:="Должность='Представитель'"
End Sub
```

Ваша первая форма в *MS Access*

Для создания формы в окне базы данных:

- ❒ нажмите кнопку **Формы;**
- ❒ нажмите кнопку **Создание формы в режиме конструктора;**
- ❒ откроется новая форма в режиме **Конструктор**. Кроме того, отобразится **Панель инструментов** и окно свойств формы (рис. 18.2). Если этих элементов нет, то выберите соответствующие команды из меню **Вид;**

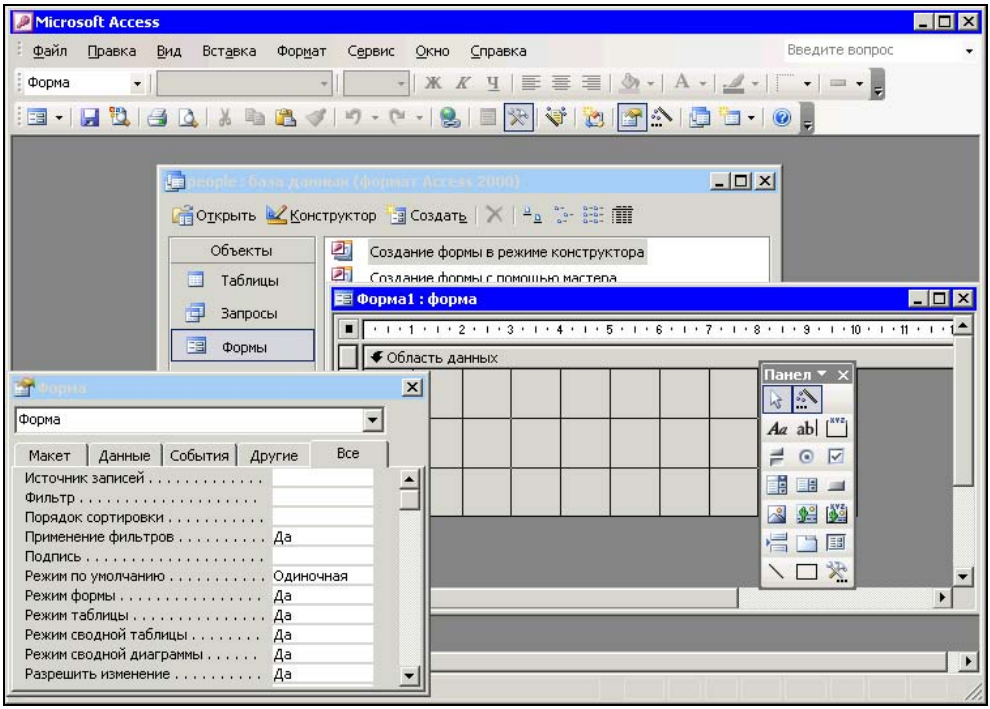


Рис. 18.2. Форма в режиме **Конструктор**

- используя вкладку **Макет** окна свойств, установите значение свойств формы, как показано в табл. 18.1;

Таблица 18.1. *Значения свойств формы, установленные в окне свойств*

Свойство	Значение
`Подпись`	`Моя первая форма`
`Кнопки перехода`	`Нет`
`Разделительные линии`	`Нет`
`Область выделения`	`Нет`

- добавьте на форму кнопку из панели инструментов. Если после добавления кнопки в форму появится окно мастера **Создание кнопок**, нажмите кнопку **Отмена**. Конечно, кнопки можно конструировать и при помощи мастера, но, делая это вручную, можно добиться более тонкой настройки;
- используя вкладку **Все** окна свойств, установите значения свойств кнопки `Подпись` и `Имя` равными `Привет` и `cmdHello`;

- для создания обработчика событий из контекстного меню кнопки выберите команду **Обработка событий**. В открывшемся окне **Построитель** выберите **Программы** и нажмите кнопку **OK**;
- откроется окно редактора кода с созданной мастером проекта процедурой обработки события `Click` объекта `cmdHello`. В нее добавьте только одну инструкцию, как показано в листинге 18.3, которая отобразит окно с приветствием при нажатии кнопки;

Листинг 18.3. Ваша первая форма в *MS Access*

```
Option Compare Database

Private Sub cmdHello_Click()
   MsgBox "Привет, всем!"
End Sub
```

- для тестирования формы закройте окно редактора кода. Закройте также окно конструктора формы. При этом отобразится окно, предлагающее сохранить созданную форму. Нажмите кнопку **Да**. На экране отобразится окно **Сохранение**. В поле **Имя формы** введите `Приветствие` и нажмите кнопку **OK**. В окно базы данных в группу **Формы** добавится новый элемент **Приветствие**. Для тестирования формы просто щелкните на этом элементе (рис. 18.3).

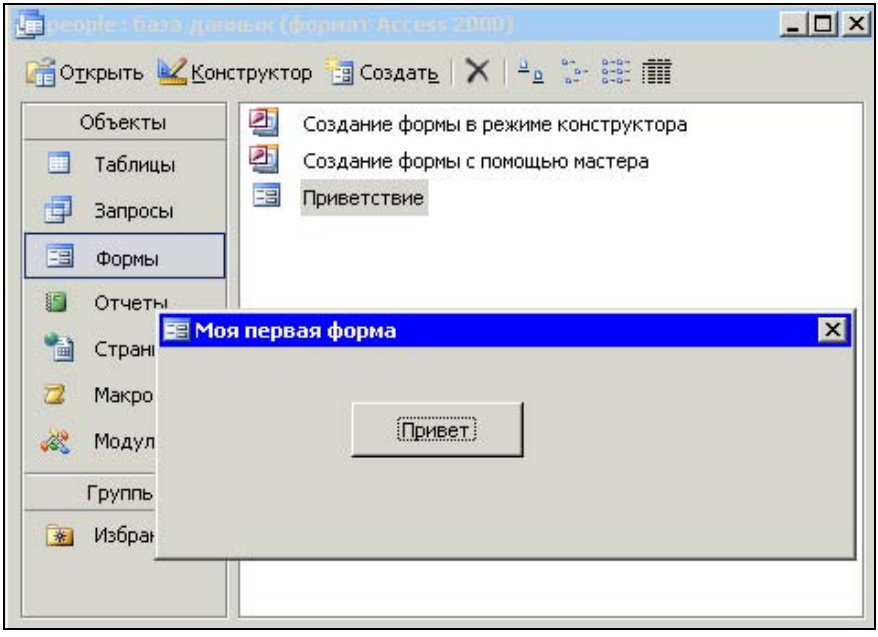


Рис. 18.3. Ваша первая форма в *MS Access*

Создание всплывающей формы

В MS Access легко создать всплывающую форму, т. е. форму, которая отображается при открытии базы данных. Для этого MS Access предоставляет в распоряжение разработчика средства контроля над действиями, выполняемыми при открытии базы данных. Итак, сконструируем всплывающую форму с фоновым рисунком и надписью (рис. 18.4).

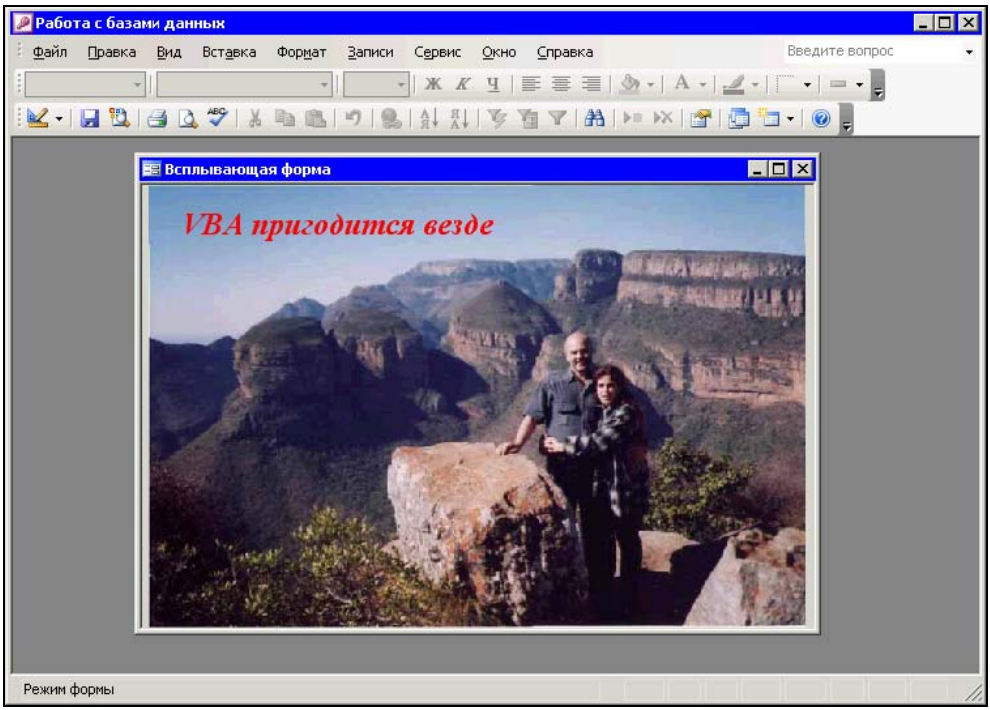


Рис. 18.4. Всплывающая форма с фоновым рисунком и надписью

- Добавьте в базу данных форму.
- Перейдите в режим **Конструктор**.
- Добавьте на форму надпись для вывода текста.
- Используя окно свойств, установите значения свойств, как показано в табл. 18.2, формы с тем, чтобы создать фоновый рисунок, а надписи — с тем чтобы отформатировать отображаемый текст.

Таблица 18.2. *Значения свойств формы и надписи, установленные в окне свойств*

Объект	Свойство	Значение
Форма	Рисунок	Ссылка на файл с растровым рисунком, который будет исполнять роль фонового рисунка
	Выравнивание рисунка	По центру
	Масштабы рисунка	По размеру
	Подпись	Всплывающая форма
	Кнопки перехода	Нет
	Разделительные линии	Нет
	Область выделения	Нет
Надпись	Подпись	VBA пригодится везде
	Тип фона	Прозрачный
	Цвет текста	255
	Размер шрифта	18
	Курсив	Да
	Насыщенность	Сверхжирный

- Закройте окно конструктора формы и присвойте форме имя `Splash`.
- Для того чтобы форма отображалась при открытии базы данных, выберите команду **Сервис | Параметры запуска**. В окне **Параметры запуска** (рис. 18.5) в списке **Вывод формы/страницы** выберите `Splash`, а в поле **Заголовок приложения** введите `Работа с базами данных`. Нажмите кнопку **OK**.

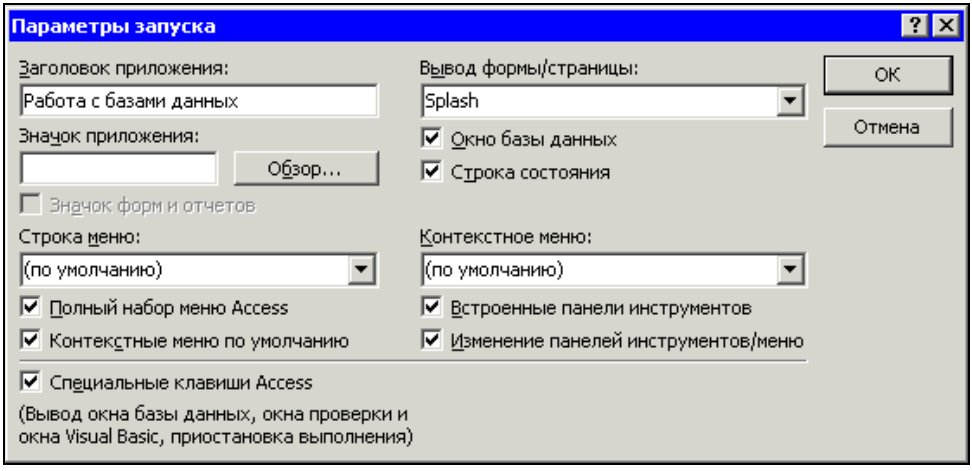


Рис. 18.5. Окно **Параметры запуска**

Обращение к объекту по имени

Один из способов обращения к объекту — это указание его места в иерархии, т.е. последовательность всех имен объектов и семейств, начиная с имени главного объекта в иерархии. Для разделения имен применяются два специальных символа: символ "!" используется для перехода от семейства к одному из его элементов, а символ "." применяется для перехода либо к семейству, либо объекту, производному от данного объекта. Например, следующая строка возвращает значение свойства `Value` поля `txtTarget`, расположенного на форме `frmFirst`.

```
Forms!frmFirst.txtTarget.Value
```

В качестве примера создадим простое приложение, которое передает значение из одной формы в другую. Итак, создайте две формы:

- на первой форме расположите поле и кнопку. Установите значения свойств `Имя` формы, поля и кнопки равными `frmFirst`, `txtTarget` и `cmdSend`;
- на второй форме расположите поле. Установите значения свойств `Имя` формы и поля равными `frmSecond` и `txtTarget`.

Нам необходимо добиться, чтобы при нажатии кнопки на первой форме открывалась вторая, причем в ее поле отображалось значение из поля первой формы. Для этого в модули форм достаточно добавить следующий код (листинги 18.4, *а*, *б*).

Листинг 18.4, *а*. Передача значения из одной формы в другую

```
Private Sub cmdSend_Click()
   DoCmd.OpenForm FormName:="frmSecond", _
                  View:=acNormal, WindowMode:=acDialog
End Sub
```

Листинг 18.4, *б*. Передача значения из одной формы в другую

```
Private Sub Form_Load()
   txtTarget.Value = Forms!frmFirst!txtSource.Value
End Sub
```

Источники данных

Формы и отчеты делятся на две категории — *связанные* или *несвязанные* (*свободные*). При создании новой формы или отчета указывается значение свойства `Источник записей` (`RecordSource`). Если значение этого свойства не задано, то форма или отчет считаются несвязанными. В качестве значения этого

свойства можно указать имя таблицы, хранимого запроса или оператора SQL, возвращающего набор записей.

Если форма и отчет связаны с источником данных, то можно связать элементы управления в пределах этой формы или отчета с конкретным полем источника данных, указав имя этого поля в качестве значения свойства `Данные` (`ControlSource`). Установки значений свойств можно сделать как при помощи окна свойств, так и в коде. В качестве примера рассмотрим базу данных people.mdb, состоящую из одной таблицы **Народ** и имеющую четыре поля: **ID** (тип — счетчик), **Фамилия** (тип — текстовый), **Рост** (тип — число) и **ДатаРождения** (тип — дата/время). Создайте форму, на которой расположите три поля (рис. 18.6). При помощи окна свойств установите им значения свойства `Имя` равными `txtName`, `txtHeight` и `txtBirthday`, а значения свойства `Подпись` — `Фамилия`, `Рост` и `Дата Рождения`. Назовите форму **Браузер**. В модуле формы наберите следующий код (листинг 18.5). Простейший браузер просмотра таблицы базы данных готов.

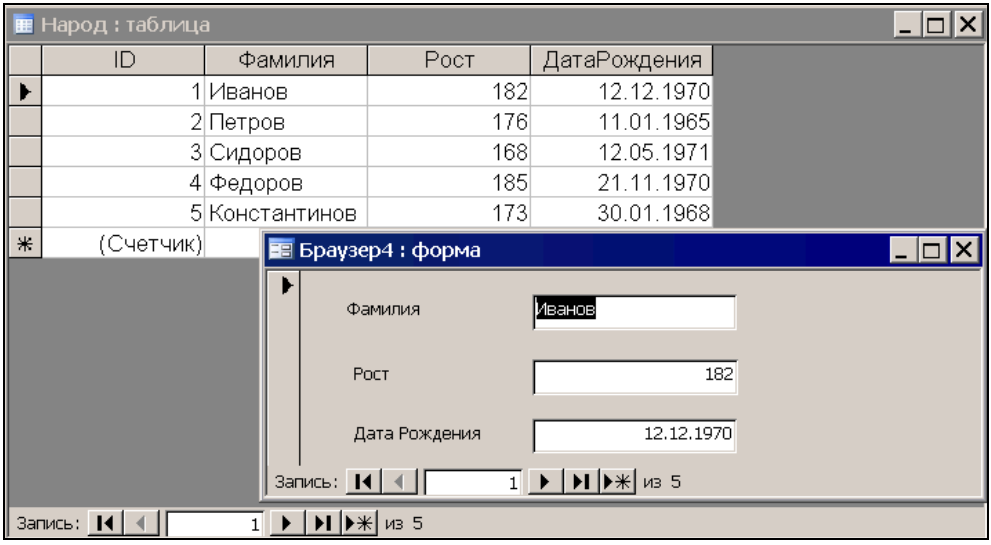


Рис. 18.6. Связанные форма и элементы управления

Листинг 18.5. Связанные форма и элементы управления

```
Private Sub Form_Load()
   Me.RecordSource = "Народ"
   txtName.ControlSource = "Фамилия"
   txtHeight.ControlSource = "Рост"
   txtBirthday.ControlSource = "ДатаРождения"
End Sub
```

Извлечение данных из списка в несвязанный элемент управления

Список заполняется на основе установки в качестве значения свойства Источник строк (RowSource) значения SQL выражения. Свойства Число столбцов (ColumnCount), Ширина столбцов (ColumnWidths) позволяют задавать число отображаемых столбцов таблицы и их ширину. Для того чтобы из списка извлечь данные в несвязанные элементы управления, надо установить в качестве значения их свойства Данные (ControlSource) ссылку на соответствующие столбцы списка, причем обратите внимание, что нумерация столбцов начинается с нуля. Продемонстрируем, как это делается, на примере (рис. 18.7), когда в список выводится список всех фамилий таблицы **Народ**. При выборе элемента из этого списка в поля **Дата Рождения** и **Рост** выводятся соответствующие пояснительные данные. Добавьте в базу данных еще одну форму, на которой расположите список и два поля. При помощи окна свойств установите форме и элементам управления значения свойств, как показано в табл. 18.3. Проект готов.

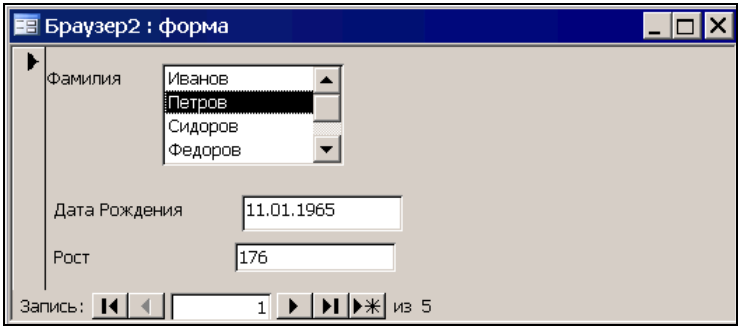


Рис. 18.7. Извлечение данных из списка

Таблица 18.3. *Значения свойств формы и надписи, установленные в окне свойств*

Объект	Свойство	Значение
Форма	Источник записей	Народ
Список	Имя	lstName
	Подпись	Фамилия
	Источник строк	SELECT Народ.ID, Народ.Фамилия, Народ.Рост, Народ.ДатаРождения FROM Народ;
	Число столбцов	4
	Ширина столбцов	0 см;2,545 см;0 см;0 см

Таблица 18.3 *(окончание)*

Объект	Свойство	Значение
Поле	Имя	Дата Рождения
	Подпись	txtBirthday
	Данные	=lstName.column(1)
Поле	Имя	txtHeight
	Подпись	Рост
	Данные	=lstName.column(2)

Программная навигация по записям

Навигацию по записям можно осуществлять как связывая элементы управления, так и программно. В качестве примера программной навигации добавьте на форму из раздела *Источники данных* четыре кнопки, причем расположите их в разделе примечаний формы (рис. 18.8). Если это раздел не отображается, воспользуйтесь командой контекстного меню **Заголовок/примечание формы**. При помощи окна свойств установите значения свойств, как показано в табл. 18.4.

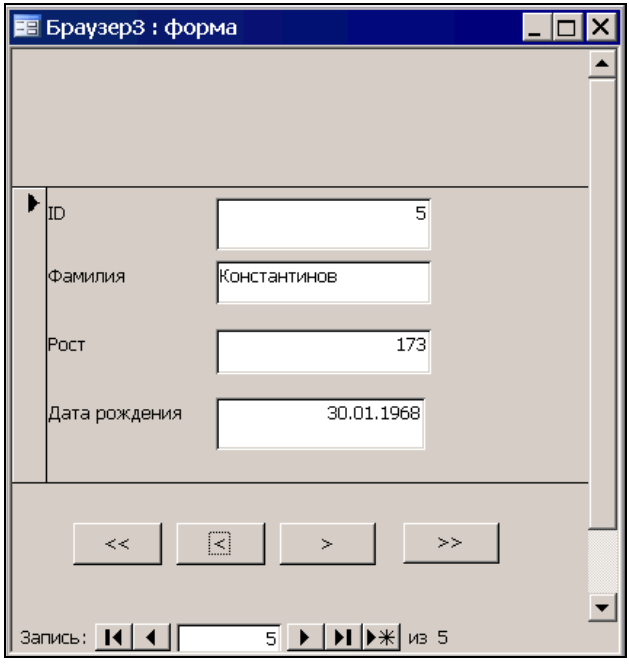


Рис. 18.8. Программная навигация по записям

Таблица 18.4. *Значения свойств формы и кнопок, установленные в окне свойств*

Объект	Свойство	Значение
Форма	Источник записей	Народ
Кнопка	Имя	cmdFirst
	Подпись	<<
Кнопка	Имя	cmdPrevious
	Подпись	<
Кнопка	Имя	cmdNext
	Подпись	>
Кнопка	Имя	cmdLast
	Подпись	>>

Для того чтобы код навигации был универсальным для всех форм, создайте модуль, в который введите код из листинга 18.6, *а*, а в модуле формы введите код из листинга 18.6, *б*. Программно навигация по записям реализуется методом `GoToRecord` объекта `DoCmd`. Для того чтобы при попытке перемещения за пределы первой и последней записи не генерировалась ошибка, в код добавлен обработчик ошибок.

Листинг 18.6, *а*. Навигация по записям. Стандартный модуль

```
Public Sub FirstRecord()
   On Error GoTo errorhandler
   DoCmd.GoToRecord Record:=acFirst
errorhandler:
End Sub

Public Sub PreviousRecord()
   On Error GoTo errorhandler
   DoCmd.GoToRecord Record:=acPrevious
errorhandler:
End Sub

Public Sub NextRecord()
   On Error GoTo errorhandler
   DoCmd.GoToRecord Record:=acNext
errorhandler:
```

```
End Sub

Public Sub LastRecord()
   On Error GoTo errorhandler
   DoCmd.GoToRecord Record:=acLast
errorhandler:
End Sub
```

Листинг 18.6, *б*. Навигация по записям. Модуль формы

```
Private Sub cmdFirst_Click()
   FirstRecord
End Sub

Private Sub cmdPrevious_Click()
   PreviousRecord
End Sub

Private Sub cmdNext_Click()
   NextRecord
End Sub

Private Sub cmdLast_Click()
   LastRecord
End Sub
```

Создание документов в Word на основе данных, хранящихся в базе данных

В качестве автоматизации работы в Access расширим функциональность формы **Браузер** из раздела *Источники данных*. На этой форме будет дополнительно расположена кнопка **Письмо** (рис. 18.9), при нажатии на которую на основе шаблона letter.dot (рис. 18.10) и текущей записи будет создаваться уведомительное письмо указанному гражданину (рис. 18.11).

Для реализации данного проекта необходимо первоначально в Word создать шаблон документа, в котором в тех местах, где требуется вывести данные из базы данных, надо разместить ключевые слова. В данном случае это `{Name},{Birthday}` и `{Height}`. Кроме того, в базе данных надо создать таблицу **Замена,** у которой имеются два поля — **ЗаменяемыйКод** и **ТекстЗамены**, в первом

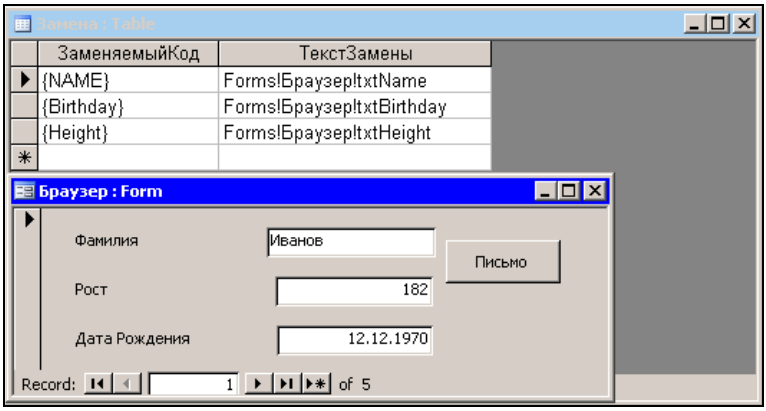


Рис. 18.9. Форма **Браузер** и таблица **Замена**

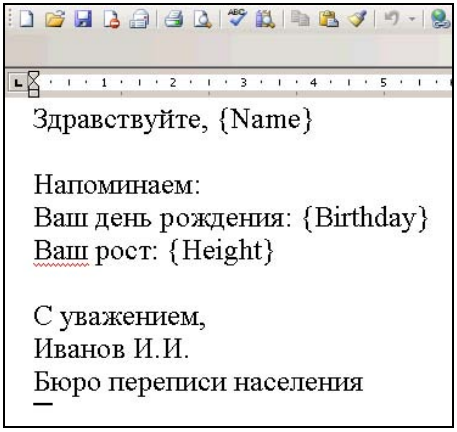


Рис. 18.10. Шаблон письма

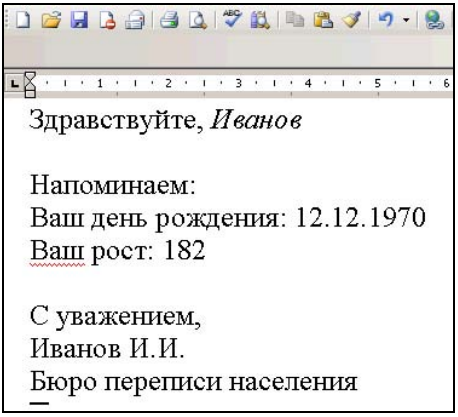


Рис. 18.11. Созданное письмо

из которых выводится список ключевых слов, а во втором — откуда подставляется в них текст. Конечно, на форму надо добавить кнопку, со значениями свойств `Имя` и `Подпись` равными `cmdLetter` и `Письмо`, а в модуле формы набрать код из листинга 18.7.

Листинг 18.7. Создание письма в Word на основе данных, хранимых в базе данных

```
Private Sub cmdLetter_Click()
   Dim rstReplace As New ADODB.Recordset
   Dim strPath As String
   Dim strFullName As String
   Dim strReplaceWith As Variant
   Dim objWord As Word.Application
   Dim objDoc As Word.Document
   strPath = CurrentProject.Path
   strFullName = strPath & "\letter.dot"
   Set objWord = New Word.Application

   Set objDoc = objWord.Documents.Add(Template:=strFullName)
   objWord.Visible = True
   rstReplace.Open "Замена", CurrentProject.Connection
   Do While Not rstReplace.EOF
      strReplaceWith = Eval(rstReplace!ТекстЗамены)
      strReplaceWith = IIf(IsNull(strReplaceWith), _
                           "  ", CStr(strReplaceWith))
      With objDoc.Content.Find
        If rstReplace!ЗаменяемыйКод = "{Name}" Then
           With .Replacement
              .ClearFormatting
              .Font.Italic = True
           End With
        End If
        .Execute FindText:=rstReplace!ЗаменяемыйКод, _
                 ReplaceWith:=strReplaceWith, _
                 Format:=True, _
                 Replace:=wdReplaceAll
      End With
      rstReplace.MoveNext
```

```
    Loop
    Exit Sub
errorhandler:
    MsgBox "При создании письма произошел сбой"
End Sub
```

Построение отчета

В Access имеются мощные средства построения отчетов вручную. Но их можно также построить и в коде. Код из листинга 18.8 служит примером создания подобного отчета. В нем на основе таблицы **Товары** базы данных **Борей** создается отчет о наличии товаров на складе, а именно, в виде отчета выводятся поля **КодТовара**, **Марка**, **Цена**, **НаСкладе**. Отчет имеет простейшую структуру, а именно, в верхнем колонтитуле в надписи выведены названия полей, а в области данных в поля отображены соответствующие записи (рис. 18.12).

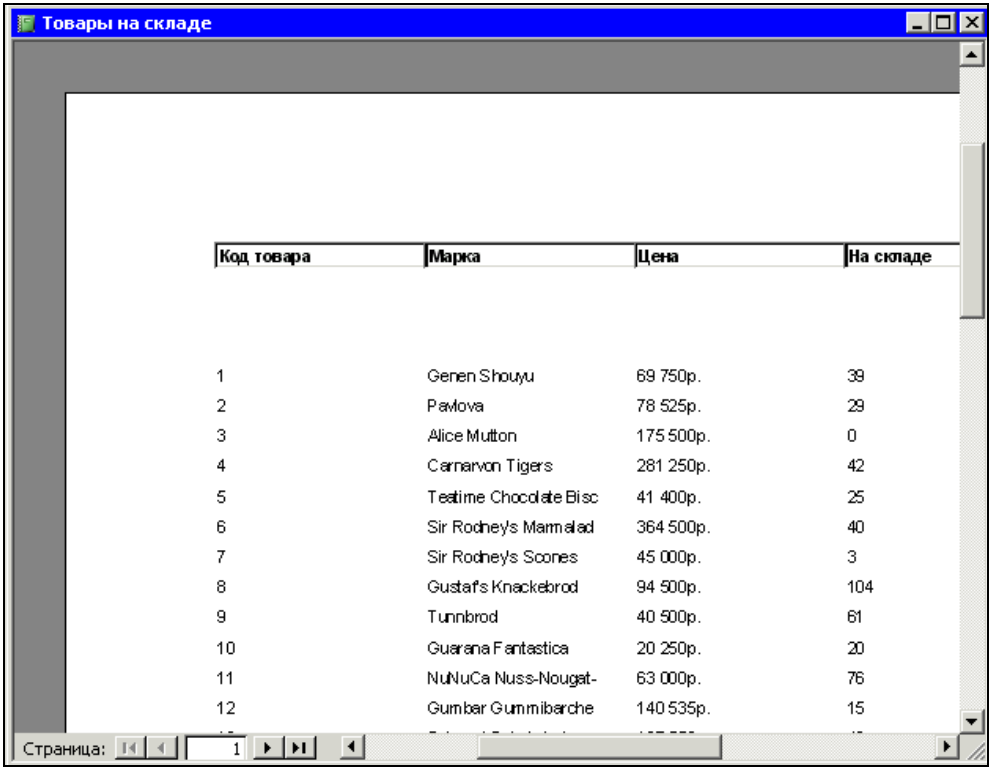
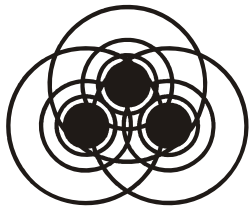


Рис. 18.12. Построенный отчет

Листинг 18.8. Построение отчета

```
Public Sub ReportBuilder()
   Dim rpt As Report
   Dim tb As Access.TextBox
   Dim lb As Access.Label
   Dim rst As ADODB.Recordset
   Dim fld As ADODB.Field
   Dim LeftPos As Long, TopPos As Long

   LeftPos = 0: TopPos = 0
   Set rpt = CreateReport
   rpt.RecordSource = " Товары"
   rpt.Caption = "Товары на складе"
   Set rst = New ADODB.Recordset
   rst.Open _
      Source:="Select КодТовара, Марка, Цена, НаСкладе From Товары", _
      ActiveConnection:=CurrentProject.Connection, _
      CursorType:=adOpenForwardOnly
   For Each fld In rst.Fields
      Set lb = CreateReportControl(ReportName:=rpt.Name, _
                                   ControlType:=acLabel, _
                                   Section:=acPageHeader, _
                                   ColumnName:=fld.Name, _
                                   Left:=LeftPos, _
                                   Top:=TopPos)
      lb.FontBold = True
      lb.Width = 2000
      lb.SpecialEffect = 2
      LeftPos = LeftPos + 2000
   Next
   LeftPos = 0
   For Each fld In rst.Fields
      Set tb = CreateReportControl(ReportName:=rpt.Name, _
                                   ControlType:=acTextBox, _
                                   Section:=acDetail, _
                                   ColumnName:=fld.Name, _
                                   Left:=LeftPos, _
                                   Top:=TopPos)
      tb.TextAlign = 1
      LeftPos = LeftPos + 2000
   Next
   rpt.Section(acDetail).Height = 1.2 * tb.Height
   DoCmd.OpenReport rpt.Name, acViewPreview
End Sub
```

Глава 19



VBA и анализ данных

Обработка данных: сортировка, фильтрация, сводные таблицы, списки

MS Excel предоставляет в распоряжение разработчика весьма эффективные средства по обработке и анализу данных, а именно, по их консолидации, сортировке и фильтрации, работе со списками, подведению промежуточных итогов, конструированию сценариев и структур, а также созданию сводных таблиц и диаграмм. Применение же соответствующих объектных моделей обеспечивает создание полностью автоматизированных проектов как по обработке и анализу данных, так и по принятию решения.

Сортировка данных

Сортировка позволяет выстраивать данные в алфавитном или цифровом порядке по возрастанию или убыванию. MS Excel может сортировать строки списков и баз данных, а также столбцы рабочих листов. При сортировке текста в таблице можно сортировать как один столбец, так и таблицу целиком. Кроме того, можно выполнить сортировку по нескольким словам или полям в одном столбце таблицы. Например, если столбец содержит имена и фамилии, то можно сортировать его по имени или по фамилии, аналогично тому, как если бы имена и фамилии были в списке, а не в таблице.

Сортировка по выделенному полю

MS Excel позволяет сортировать список по выделенному полю. Для этого:

1. Выберите ячейку в сортируемом списке. MS Excel сам распознает список по выделенному полю.

2. Нажмите кнопку **Сортировать по возрастанию** или **Сортировать по убыванию**.

Если MS Excel распознает при сортировке по выделенному полю не тот диапазон (фрагмент списка), который следует отсортировать, то первоначально надо выделить искомый диапазон, а затем произвести сортировку. При этом она будет осуществляться по первому столбцу из выделенного диапазона.

Метод *Sort*

Метод `Sort` осуществляет сортировку данных с учетом до трех критериев, по которым производится сортировка. Метод `Sort` позволяет сортировать строки списков, сводных таблиц и баз данных, а также столбцы рабочих листов. Сортировка данных на рабочем листе производится пользователем командой **Данные | Сортировка**.

`expression.Sort(`*`Key1`*`, `*`Order1`*`, `*`Key2`*`, `*`Type`*`, `*`Order2`*`, `*`Key3`*`, `*`Order3`*`, `*`Header`*`, `*`OrderCustom`*`, `*`MatchCase`*`, `*`Orientation`*`, `*`SortMethod`*`, `*`DataOption1`*`, `*`DataOption2`*`, `*`DataOption3`*`)`, где:

- *`expression`* — ссылка на ячейку диапазона или на сам диапазон, который будет сортироваться;
- *`Key1`* — необязательный параметр, задающий ссылку на первое упорядочиваемое поле;
- *`Order1`* — необязательный параметр, определяющий порядок упорядочивания поля, заданного параметром *`Key1`*. Допустимыми значениями являются следующие константы `XlSortOrder`: `XlAscending` (возрастающий порядок), `xlDescending` (убывающий);
- *`Key2`* — необязательный параметр, задающий ссылку на второе упорядочиваемое поле;
- *`Type`* — необязательный параметр, задающий элементы, которые должны быть отсортированы. Используется только со сводными таблицами;
- *`Order2`* — необязательный параметр, определяющий порядок упорядочивания поля, заданного параметром *`Key2`*. Допустимыми значениями являются константы `XlSortOrder`;
- *`Key3`* — необязательный параметр, задающий ссылку на третье упорядочиваемое поле;
- *`Order3`* — необязательный параметр, определяющий порядок упорядочивания поля, заданного параметром *`Key3`*. Допустимыми значениями являются константы `XlSortOrder`;
- *`Header`* — необязательный параметр, определяющий, имеется ли в первой строке списка заголовок. Допустимыми значениями являются следующие

константы `XlYesNoGuess`: `xlYes` (первая строка диапазона содержит заголовок, который не сортируется), `xlNo` (первая строка диапазона не содержит заголовок, по умолчанию считается данное значение), `xlGuess` (MS Excel решает сам, имеет ли список заголовок);

- *`OrderCustom`* — необязательный параметр, задающий пользовательский порядок сортировки. Является целым числом, указывающим порядковый номер списка, используемого в качестве шаблона сортировки;
- *`MatchCase`* — необязательный параметр, указывающий, надо ли учитывать регистры букв при сортировке;
- *`Orientation`* — необязательный параметр, специфицирующий ориентацию сортировки. Допустимыми значениями являются следующие константы `XlSortOrientation`: `xlTopToBottom` (сортировка осуществляется сверху вниз, т. е. по строкам), `xlLeftToRight` (сортировка осуществляется слева направо, т. е. по столбцам);
- *`SortMethod`* — необязательный параметр, указывающий метод сортировки. Применяется для таких языков, как китайский, японский;
- *`DataOption1`* — необязательный параметр, специфицирующий, как должен сортироваться текст в поле, заданном параметром *`Key1`*. Допустимыми значениями являются следующие константы `XlSortDataOption`: `xlSortTextAsNumbers` (числовые и текстовые данные сортируются вместе), `xlSortNormal` (числовые и текстовые данные сортируются по раздельности);
- *`DataOption2`* — необязательный параметр, специфицирующий, как должен сортироваться текст в поле, заданном параметром *`Key2`*. Допустимыми значениями являются константы `XlSortDataOption`.
- *`DataOption3`* — необязательный параметр, специфицирующий, как должен сортироваться текст в поле, заданном параметром *`Key3`*. Допустимыми значениями являются константы `XlSortDataOption`.

Пример приложения, сортирующего данные

Рассмотрим пример простого приложения, в котором автоматизирован как процесс сортировки, так и процесс возвращения данных в исходный порядок. Сортировать будем данные таблицы **Сотрудники** базы данных **Борей** по странам и фамилиям. Итак, создайте книгу, состоящую из одного рабочего листа **Сотрудники**, в который импортируйте поля **КодСотрудника**, **Фамилия**, **Должность**, **Город**, **Страна**, **ДомашнийТелефон** таблицы **Сотрудники** базы данных **Борей**. После этого в стандартный модуль и модуль **ЭтаКнига** надо добавить приводимый ниже код (листинги 19.1, *а*, *б*).

Листинг 19.1, *а*. Сортировка данных из таблицы Сотрудники. Стандартный модуль

```
Public cb As CommandBarComboBox

Sub DoSort()
   Select Case cb.ListIndex
      Case 1
         Range("A1").Select
         Selection.Sort Key1:=Range("A1"), _
                        Order1:=xlAscending, Header:=xlYes
      Case 2
         Range("A1").Select
         Selection.Sort Key1:=Range("E2"), Order1:=xlAscending, _
                        Key2:=Range("B2"), Order2:=xlAscending, _
                        Header:=xlYes
   End Select
   Range("A1").Select
End Sub
```

Листинг 19.1, *б*. Сортировка данных из таблицы Сотрудники. Модуль ЭтаКнига

```
Private Sub Workbook_Open()
   Dim cbSort As CommandBar
   Set cbSort = Application.CommandBars.Add(Name:="Сортировка", _
                            MenuBar:=True, Temporary:=True)
   cbSort.Visible = True
   Set cb = cbSort.Controls.Add(Type:=msoControlDropdown)
   With cb
      .AddItem "Не отсортированные", 1
      .AddItem "Отсортированные", 2
      .ListIndex = 1
      .DropDownWidth = 120
      .OnAction = "DoSort"
   End With
End Sub

Private Sub Workbook_BeforeClose(Cancel As Boolean)
   Application.CommandBars("Сортировка").Delete
End Sub
```

Примечание

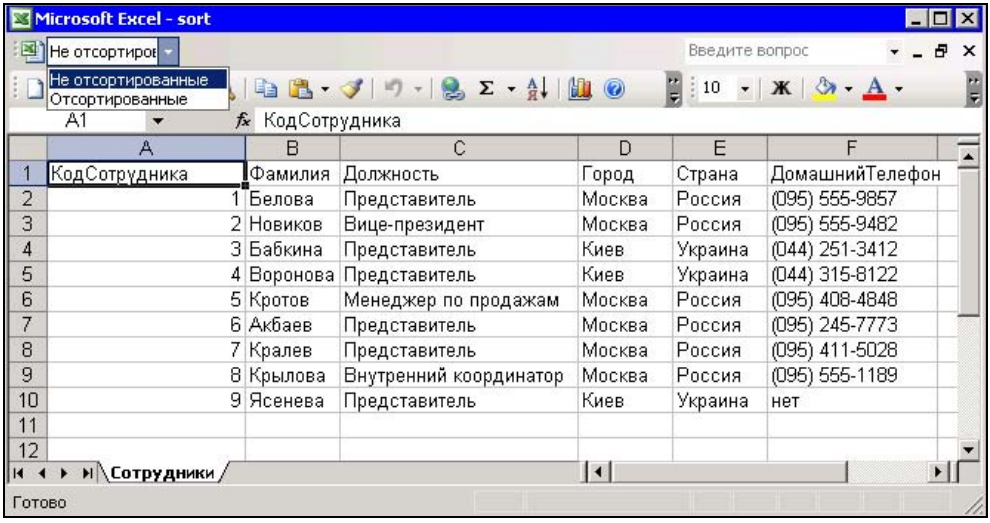
Обратите внимание, что после сортировки в коде выбирается одна ячейка. Это делается для того, чтобы снять выделение со всей таблицы.

В стандартном модуле имеется процедура и объявлена открытая переменная:

- в открытой переменной `cb` объявлен объект `CommandBarComboBox`, который инкапсулирует в себе данные о раскрывающемся списке, управляющем сортировкой из пользовательской панели инструментов;
- процедура `DoSort` в зависимости от выбранного из раскрывающегося списка элемента производит следующие действия:
  - если выбран элемент `Не отсортированные`, (значение индекса его в списке равно 1), то сортировка производится по полю **КодСотрудника**, тем самым восстанавливая первоначальную сортировку таблицы;
  - если же выбран элемент `Отсортированные`, (значение индекса его в списке равно 2), то сортировка производится по полям **Страна** и **Фамилия**.

В модуле **ЭтаКнига** имеются две процедуры, которые создают интерфейс проекта при открытии книги и его удаление при ее закрытии (рис. 19.1):

- процедура обработки события `Open` объекта `Workbook` конструирует панель инструментов **Сортировка**, на которой расположен раскрывающийся список с двумя элементами `Не отсортированные` и `Отсортированные`, которые и задают различные режимы выполнения процедуры `DoSort`;
- процедура обработки события `BeforeClose` объекта `Workbook` удаляет панель **Сортировка** при закрытии книги.



	A	B	C	D	E	F
1	КодСотрудника	Фамилия	Должность	Город	Страна	ДомашнийТелефон
2	1	Белова	Представитель	Москва	Россия	(095) 555-9857
3	2	Новиков	Вице-президент	Москва	Россия	(095) 555-9482
4	3	Бабкина	Представитель	Киев	Украина	(044) 251-3412
5	4	Воронова	Представитель	Киев	Украина	(044) 315-8122
6	5	Кротов	Менеджер по продажам	Москва	Россия	(095) 408-4848
7	6	Акбаев	Представитель	Москва	Россия	(095) 245-7773
8	7	Кралев	Представитель	Москва	Россия	(095) 411-5028
9	8	Крылова	Внутренний координатор	Москва	Россия	(095) 555-1189
10	9	Ясенева	Представитель	Киев	Украина	нет
11						
12						

Рис. 19.1. Сортировка таблицы **Товары**

Фильтрация

Фильтрация — это эффективный способ поиска подмножества данных и работы с ними в списке. В отфильтрованном списке отображаются только строки, отвечающие условиям отбора, заданным для столбца. MS Excel предоставляет две команды для фильтрации списков:

- *автофильтр*, включая фильтр по выделенному, для простых условий отбора;
- *расширенный фильтр* для более сложных условий отбора.

При фильтрации порядок записей в списке не изменяется, строки, которые не удовлетворяют критерию фильтрации, временно скрываются. Отфильтрованные строки можно редактировать, форматировать и выводить на печать, а также создавать на их основе диаграммы, не перемещая их и не изменяя порядок строк.

Использование автофильтра

При использовании команды **Автофильтр** справа от подписей столбцов в фильтруемом списке появляются стрелки автофильтра. У отфильтрованных элементов стрелки автофильтра окрашиваются в синий цвет, а у неотфильтрованных они остаются черного цвета.

В раскрывающемся списке автофильтров можно выбрать (рис. 19.2):

- команду **Все** для отображения всех элементов списка;
- команду **Первые 10**, позволяющую отобразить некоторое количество (по умолчанию 10) наибольших (по умолчанию) или наименьших элементов списка;
- команду **Условие**, чтобы вызвать диалоговое окно **Пользовательский автофильтр**, где можно задать простой критерий, содержащий до двух условий;
- значение поля для поиска точного соответствия.

	A	B	C	D	E	F	G
1	КодСотрудника	Фамилия	Имя	Должность	ДатаНайма	Город	Страна
2	1	Белова			01.05.1992	Москва	Россия
3	2	Новиков			14.08.1992	Москва	Россия
4	3	Бабкина			01.04.1992	Киев	Украина
5	4	Воронова			03.05.1993	Киев	Украина
6	5	Кротов			17.10.1993	Москва	Россия
7	6	Акбаев			17.10.1993	Москва	Россия
8	7	Кралев			02.01.1994	Москва	Россия
9	8	Крылова			05.03.1994	Москва	Россия
10	9	Ясенева	Инна	Представитель	15.11.1994	Киев	Украина

Сортировка по возрастанию
Сортировка по убыванию
(Все)
(Первые 10...)
(Условие...)
Вице-президент
Внутренний координатор
Менеджер по продажам
Представитель

Рис. 19.2. Раскрывающийся список автофильтра

Описание процесса автофильтрации

Рассмотрим процесс автофильтрации на примере данных таблицы **Сотрудники**, взятой из учебной базы данных **Борей**.

1. Перед применением автофильтра убедитесь, что он не используется для какого либо другого списка. Для этого выберите команду **Данные | Фильтр**. Если команда **Автофильтр** отмечена флажком, выберите ее, чтобы отменить автофильтр для другого списка.
2. Выделите ячейку внутри фильтруемого списка или выделите его целиком. В нашем случае, например, выделите ячейку **A2**.
3. Выберите команду **Данные | Фильтр | Автофильтр**. В результате справа от текста, отображаемого в ячейках с названиями полей в фильтруемом списке, появятся стрелки автофильтра, а сами эти ячейки превратятся в раскрывающиеся списки.
4. Раскройте список, соответствующий полю, которое следует включить в критерий. Выберите один из допустимых критериев:
   - **Все** (вывод на экран всех записей);
   - **Первые 10** (вывод на экран заданного числа или заданного процента первых или последних записей);
   - **Условие** (вывод на экран записей по условию, заданному в отображаемом диалоговом окне **Пользовательский автофильтр**);
   - **Точное значение** (вывод на экран записей, поля которых в точности совпадают с выбранным значением).

   Например, в нашем случае в списке **Должность** выберите точное значение **Представитель**.
5. Вернитесь к предыдущему шагу, если в критерий необходимо включить другое поле. Например, в списке **ДатаНайма** выберите **Условие**. На экране отобразится диалоговое окно **Пользовательский автофильтр** (рис. 19.3). Диалоговое окно **Пользовательский автофильтр** позволяет быстро задать более сложное условие, чем простое сравнение. В левом верхнем раскрывающемся списке выбирается операция сравнения (в данном случае выберите **больше или равно**), в правом — выбирается или вводится значение (введите `03.05.93`). Затем, если необходимо, выберите один из переключателей **И**, **ИЛИ** и задайте вторую операцию сравнения и значение. В данном случае выберите переключатель **И**, операцию сравнения **меньше или равно** и введите значение `15.11.94`.
6. Нажмите кнопку **ОК**. Результат автофильтрации будет немедленно отображен на экране (рис. 19.4).

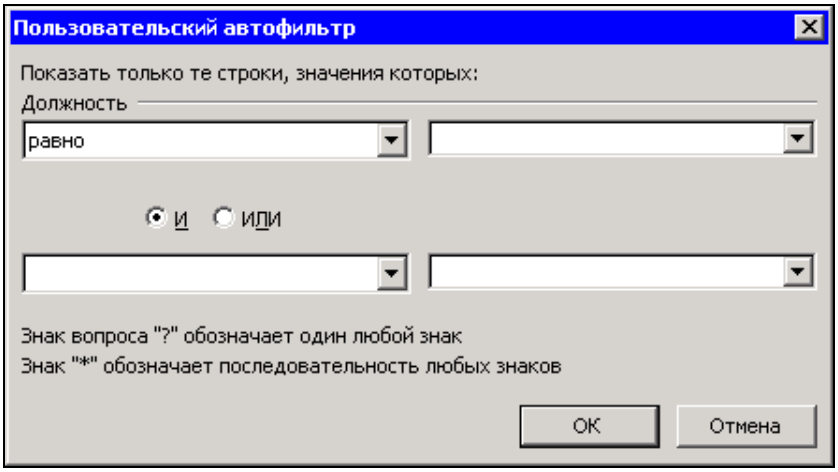


Рис. 19.3. Диалоговое окно **Пользовательский автофильтр**

	A	B	C	D	E	F	G
1	КодСотрудника	Фамилия	Имя	Должность	ДатаНайма	Город	Страна
5	4	Воронова	Дарья	Представитель	03.05.1993	Киев	Украина
7	6	Акбаев	Иван	Представитель	17.10.1993	Москва	Россия
8	7	Кралев	Петр	Представитель	02.01.1994	Москва	Россия
10	9	Ясенева	Инна	Представитель	15.11.1994	Киев	Украина
11							

Рис. 19.4. Результат автофильтрации

Метод *AutoFilter*

Метод `AutoFilter` объекта `Range` программирует выполнение команды **Данные | Фильтр | Автофильтр** для фильтрации списка по критериям, указанным в параметрах.

```
expression.AutoFilter(Field, Criteria1, Operator,
Criteria2, VisibleDropDown), где:
```

- *expression* — ссылка на ячейку диапазона или на сам диапазон, который будет фильтроваться;
- *Field* — необязательный параметр, задающий номер поля списка, в котором производится фильтрация данных. Нумерация производится с крайнего левого поля, причем первое поле имеет номер 1;
- *Criteria1* и *Criteria2* — необязательные параметры, задающие два возможных критерия фильтрации поля. Допускается использование строковой постоянной, например "101" и знаков отношений ">, < ,>=, <=, =, <>". Используйте "=" для фильтрации пустых полей, а "<>" для фильтрации

непустых полей. Если параметр опущен, то критерием является значение `All`. Если значение параметра *`Operator`* равно `xlTop10Items`, то параметр `Criteria1` определяет число отображаемых элементов (например, "`30`");

- *`Operator`* — необязательный параметр, допустимыми значениями которого могут быть следующие `XlAutoFilterOperator` константы: `xlAnd` (для логического объединения первого и второго критериев фильтрации), `xlOr` (для логического сложения первого и второго критериев), `xlTop10Items` (для отображения первых десяти элементов поля), `xlBottom10Items` (для отображения последних десяти элементов поля), `xlTop10Percent` (для отображения первых 10% элементов поля), `xlBottom10Percent` (для отображения последних 10% элементов поля);
- *`VisibleDropDown`* — необязательный параметр, принимающий логические значения. Определяет, отображаются ли раскрывающиеся списки. По умолчанию параметр принимает значение `True`.

Объекты *AutoFilter* и *Filter* и семейство *Filters*

Объект `AutoFilter` инкапсулирует в себе данные об используемом на рабочем листе автофильтре. Этот объект возвращается свойством `AutoFilter` объекта `Worksheet`. Основные свойства объектов `AutoFilter` и `Filter` приведены в табл. 19.1 и 19.2.

Таблица 19.1. *Свойства объекта* `AutoFilter`

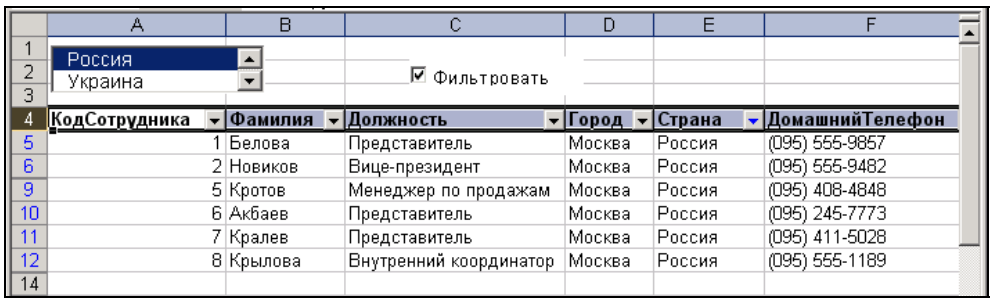
Свойство	Описание
`Filters`	Возвращает семейство `Filters` объектов `Filter`, т. е. всех фильтров, образующих искомый автофильтр. У семейства `Filters` основными свойствами являются `Count` и `Item`, которые возвращают число элементов и конкретный элемент семейства
`Range`	Возвращает диапазон, к которому применен фильтр

Таблица 19.2. *Свойства объекта* `Filter`

Свойство	Описание
`On`	Возвращает значение `True`, если фильтр установлен
`Criteria1`	Возвращает первый критерий фильтра
`Criteria2`	Возвращает второй критерий фильтра
`Operator`	Возвращает оператор фильтра. Допустимыми значениями являются константы `XlAutoFilterOperator`

Пример приложения, фильтрующего данные

Продемонстрируем применение метода `AutoFilter` на примере бизнес-ситуации создания приложения, фильтрующего по странам таблицу **Сотрудники** базы данных **Борей**. Итак, создайте книгу, состоящую из одного рабочего листа **Борей**, в который импортируйте поля **КодСотрудника**, **Фамилия**, **Должность**, **Город**, **Страна**, **ДомашнийТелефон** таблицы **Сотрудники** базы данных **Борей**, так чтобы ее верхний левый угол располагался в ячейке **A4**. Расположите на рабочем листе список и флажок. Используя окно **Properties**, установите списку значение свойства `Name` равным `lstOrder`, а флажку — значения свойств `Name` и `Caption` равными `chkFilter` и `Фильтровать`. При установленном флажке таблица будет фильтроваться по значению, выбранному в списке. При снятом флажке фильтрация отменяется (рис. 19.5). В модуле **ЭтаКнига** и модуле рабочего листа наберите код из листингов 19.2, *а*, *б*. Процедура обработки события `Open` объекта `Workbook` используется для ввода названий стран в список. В модуле рабочего листа имеются три процедуры: процедура `DoFilter`, которая и реализует фильтрацию или ее отмену, и процедуры обработки события `Click` списка и флажка, которые и вызывают процедуру `DoFilter`.



КодСотрудника	Фамилия	Должность	Город	Страна	ДомашнийТелефон
1	Белова	Представитель	Москва	Россия	(095) 555-9857
2	Новиков	Вице-президент	Москва	Россия	(095) 555-9482
5	Кротов	Менеджер по продажам	Москва	Россия	(095) 408-4848
6	Акбаев	Представитель	Москва	Россия	(095) 245-7773
7	Кралев	Представитель	Москва	Россия	(095) 411-5028
8	Крылова	Внутренний координатор	Москва	Россия	(095) 555-1189

Рис. 19.5. Фильтрация списка

Листинг 19.2, *а*. Фильтрация данных по одному выбранному сотруднику. Модуль ЭтаКнига

```
Private Sub Workbook_Open()
   Worksheets(1).lstOrder.Clear
   Worksheets(1).lstOrder.AddItem "Россия"
   Worksheets(1).lstOrder.AddItem "Украина"
   Worksheets(1).lstOrder.ListIndex = 0
End Sub
```

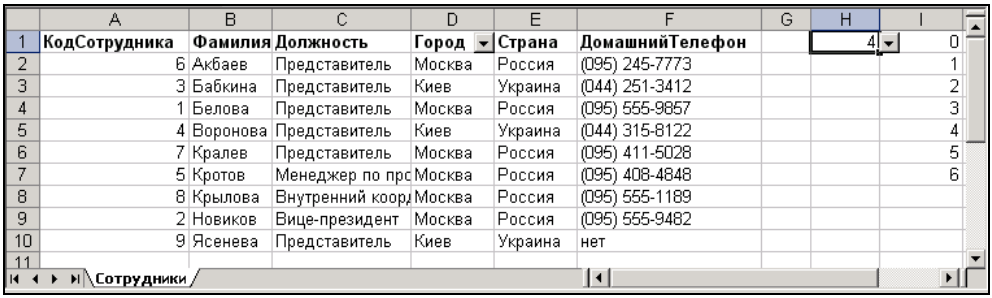
Листинг 19.2, *б*. Фильтрация данных по одному выбранному сотруднику. Модуль рабочего листа

```
Private Sub chkFilter_Click()
  DoFilter
End Sub

Private Sub lstOrder_Click()
   DoFilter
End Sub

Sub DoFilter()
   If chkFilter.Value Then
      If ActiveSheet.AutoFilterMode Then
        Rows(4).Select
        Selection.AutoFilter Field:=5, Criteria1:=lstOrder.Text
      Else
        Rows(4).Select
        Selection.AutoFilter
        Selection.AutoFilter Field:=5, Criteria1:=lstOrder.Text
      End If
    Else
      If ActiveSheet.AutoFilterMode Then
         Rows(4).Select
         Selection.AutoFilter
      End If
    End If
End Sub
```

Как сделать, чтобы в фильтре отображались только списки выбора указанных столбцов

В фильтре можно отображать не все, а только избранные списки фильтрации. Отображением списков управляет параметр `Visibledropdown` метода `AutoFilter`. В следующем примере демонстрируется его работа. В столбцы от **A** до **E** рабочего листа введены поля **КодСотрудника**, **Фамилия**, **Должность**, **Город**, **Страна**, **ДомашнийТелефон** таблицы **Сотрудники** базы данных **Борей**. В диапазон **I1:I7** введены значения от 0 до 6 (рис. 19.6). В ячейку **H1** на основе диапазона **I1:I7** вводится список значений, а в модуле рабочего листа содержится код из листинга 19.3. Создайте автофильтр. Теперь выбор значения

из списка будет вызывать то, что только поле с указанным номером будет иметь раскрывающийся список.



	A	B	C	D	E	F	G	H	I
1	КодСотрудника	Фамилия	Должность	Город	Страна	ДомашнийТелефон		4	0
2	6	Акбаев	Представитель	Москва	Россия	(095) 245-7773			1
3	3	Бабкина	Представитель	Киев	Украина	(044) 251-3412			2
4	1	Белова	Представитель	Москва	Россия	(095) 555-9857			3
5	4	Воронова	Представитель	Киев	Украина	(044) 315-8122			4
6	7	Кралев	Представитель	Москва	Россия	(095) 411-5028			5
7	5	Кротов	Менеджер по про	Москва	Россия	(095) 408-4848			6
8	8	Крылова	Внутренний коор	Москва	Россия	(095) 555-1189			
9	2	Новиков	Вице-президент	Москва	Россия	(095) 555-9482			
10	9	Ясенева	Представитель	Киев	Украина	нет			
11									

Сотрудники

Рис. 19.6. Выборочное отображение списков в фильтре

Листинг 19.3. Выборочное отображение списков в фильтре

```
Private Sub Worksheet_Change(ByVal Target As Range)
   If Target.Address = "$H$1" Then
      HideArrows Target.Value
   End If
End Sub

Sub HideArrows(n As Integer)
   Dim c As Range, r As Range
   Set r = Range(Range("A1"), Range("A1").End(xlToRight))
   Application.ScreenUpdating = False
   If n > 0 Then
      For Each c In r
         If c.Column <> n Then
           c.AutoFilter Field:=c.Column, Visibledropdown:=False
         Else
           c.AutoFilter Field:=c.Column, Visibledropdown:=True
         End If
      Next
   Else
      For Each c In r
         c.AutoFilter Field:=c.Column, Visibledropdown:=True
      Next
   End If
   Application.ScreenUpdating = True
End Sub
```

Как определить число отфильтрованных записей

Для того чтобы определить число отфильтрованных записей, можно воспользоваться свойством `SpecialCells` объекта `Range` при значении параметра равным `xlVisible`, которое возвращает число видимых ячеек специфицированного диапазона. Продемонстрируем применение этого свойства на примере. На рабочий лист из проекта предыдущего раздела добавьте кнопку. При помощи окна **Properties** установите значения ее свойств `Name` и `Caption` равными `cmdCountRecord` и `Число отобранных записей`. В модуле рабочего кода наберите код из листинга 19.4. Теперь при нажатии кнопки **Число отобранных записей** отобразится окно с информацией, как об общем числе, так и о числе отобранных записей (рис. 19.7).

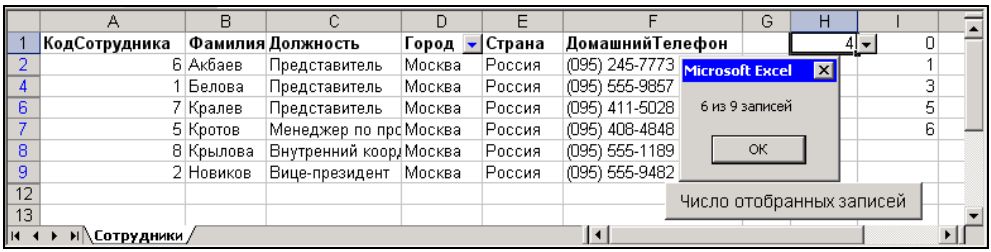


Рис. 19.7. Число отфильтрованных записей

Листинг 19.4. Число отфильтрованных записей

```
Private Sub cmdCountRecord_Click()
   Dim r As Range
   Set r = ActiveSheet.AutoFilter.Range
   MsgBox Range("A1").CurrentRegion.Columns(1). _
          SpecialCells(xlVisible).Count - 1 _
          & " из " & Range("A1").CurrentRegion.Rows.Count - 1 _
          & " записей"
End Sub
```

Сводная таблица

Сводные таблицы являются одним из наиболее мощных средств MS Excel по анализу баз данных, помещенных в таблицы или списки. Сводная таблица не просто группирует и обобщает данные, но и дает возможность провести глубокий анализ имеющейся информации. Создавая сводную таблицу, пользователь задает имена полей, которые размещаются в ее строках и столбцах.

Допускается также задание поля страницы, которое позволяет работать со сводной таблицей, как со стопкой листов. Сводные таблицы удобны при анализе данных по нескольким причинам:

- позволяют создавать обобщающие таблицы, которые дают возможность группировать однотипные данные, подводить итоги, находить статические характеристики записей;
- легко преобразуются;
- позволяют производить автоматический отбор информации;
- на основе сводных таблиц строятся диаграммы, которые динамически перестраиваются вместе с изменениями сводной таблицы.

В последующих разделах на примере таблицы заказов, взятой из базы данных **Борей** и показанной на рис. 19.8, продемонстрируем, как создаются сводные таблицы и какие задачи можно решать с их помощью.

	A	B	C	D	E
1	КодЗаказа	ДатаРазмещения	ДатаИсполнения	СтоимостьДоставки	СтранаПолучателя
2	10248	04.авг.94	16.авг.94	32,38	Финляндия
3	10249	05.авг.94	10.авг.94	11,61	Германия
4	10250	08.авг.94	12.авг.94	65,83	Бразилия
5	10251	08.авг.94	15.авг.94	41,34	Франция
6	10252	09.авг.94	11.авг.94	51,3	Бельгия
7	10253	10.авг.94	16.авг.94	58,17	Бразилия
8	10254	11.авг.94	23.авг.94	22,98	Швейцария
9	10255	12.авг.94	15.авг.94	148,33	Швейцария
10	10256	15.авг.94	17.авг.94	13,97	Бразилия
11	10257	16.авг.94	22.авг.94	81,91	Венесуэлла
12	10258	17.авг.94	23.авг.94	140,51	Австрия
13	10259	18.авг.94	25.авг.94	3,25	Мексика
14	10260	19.авг.94	29.авг.94	55,09	Германия
15	10261	19.авг.94	30.авг.94	3,05	Бразилия
16	10262	22.авг.94	25.авг.94	48,29	США
17	10263	23.авг.94	31.авг.94	146,06	Австрия
18	10264	24.авг.94	23.сен.94	3,67	Швеция
19	10265	25.авг.94	12.сен.94	55,28	Франция
20	10266	26.авг.94	31.авг.94	25,73	Финляндия
21	10267	29.авг.94	06.сен.94	208,58	Германия
22	10268	30.авг.94	02.сен.94	66,29	Венесуэлла
23	10269	31.авг.94	09.сен.94	4,56	США
24	10270	01.сен.94	02.сен.94	136,54	Финляндия
25	10271	01.сен.94	30.сен.94	4,54	США
26	10272	02.сен.94	06.сен.94	98,03	США
27	10273	05.сен.94	12.сен.94	76,07	Германия
28	10274	06.сен.94	16.сен.94	6,01	Франция
29	10275	07.сен.94	09.сен.94	26,93	Италия
30	10276	08.сен.94	14.сен.94	13,84	Мексика
31	10277	09.сен.94	13.сен.94	125,77	Германия

Заказы

Рис. 19.8. Таблица **Заказы**

Подведение итогов по странам

Создадим сводную таблицу, которая определяет суммарную стоимость доставки заказов для каждой страны.

1. Выберите команду **Данные | Сводная таблица**. На экране появится первое окно мастера сводных таблиц (рис. 19.9).

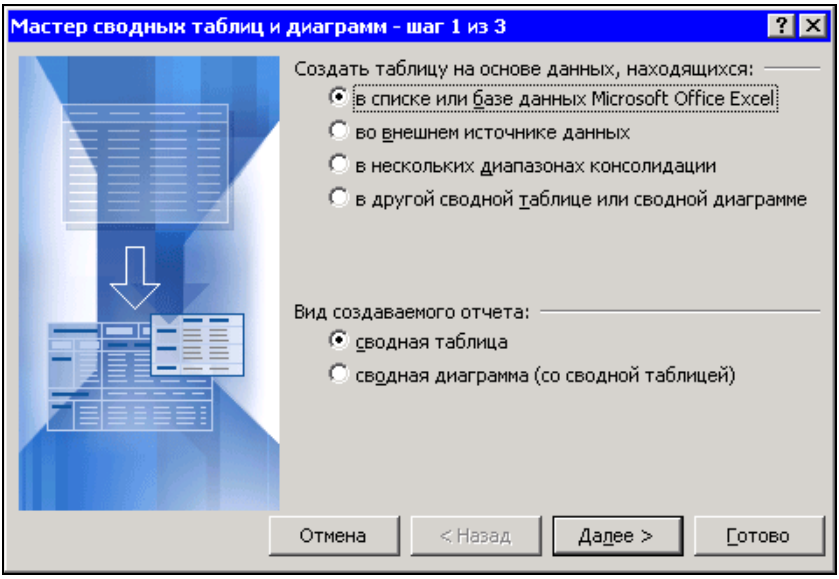


Рис. 19.9. Первое окно **Мастер сводных таблиц**

2. В первом окне мастера сводных таблиц под заголовком **Создать таблицу на основе данных, находящихся** надо указать источник данных для создания сводной таблицы. Возможны четыре источника данных, которые выбираются при помощи одного из переключателей (табл. 19.3).

Таблица 19.3. *Переключатели источников данных*

Переключатель	Источник данных
в списке или базе данных Microsoft Excel	Список или таблица с помеченными столбцами, расположенная на рабочем листе
во внешнем источнике данных	Файлы и таблицы, созданные другими программами, например, Access
в нескольких диапазонах консолидации	Несколько списков или таблица с помеченными столбцами, расположенная на рабочем листе Excel
в другой сводной таблице или сводной диаграмме	Другая существующая в активной рабочей книге сводная таблица или сводная диаграмма

В нашем случае выберите переключатель **в списке или базе данных Microsoft Excel**.

Кроме того, при помощи переключателей под заголовком **Вид создаваемого отчета** можно задать вид сводной таблицы — просто сводная таблица или сводная диаграмма со сводной таблицей. В нашем случае выберите переключатель **сводная таблица**. Нажмите кнопку **Далее**.

3. На экране появится второе окно мастера сводных таблиц (рис. 19.10). На этом шаге вы должны указать диапазон, содержащий данные, по которым будет строиться сводная таблица. Если источник данных находится в другой рабочей книге, то необходимо воспользоваться кнопкой **Обзор**. Итак, после ввода ссылки на диапазон данных, в данном случае `$A$1:$E$31`, нажмите кнопку **Далее**. На экране появится третье окно мастера сводных таблиц (рис. 19.11).

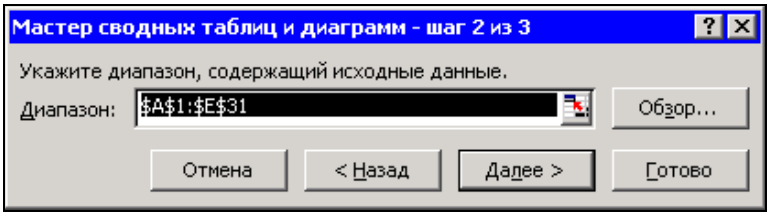


Рис. 19.10. Второе окно мастера сводных таблиц

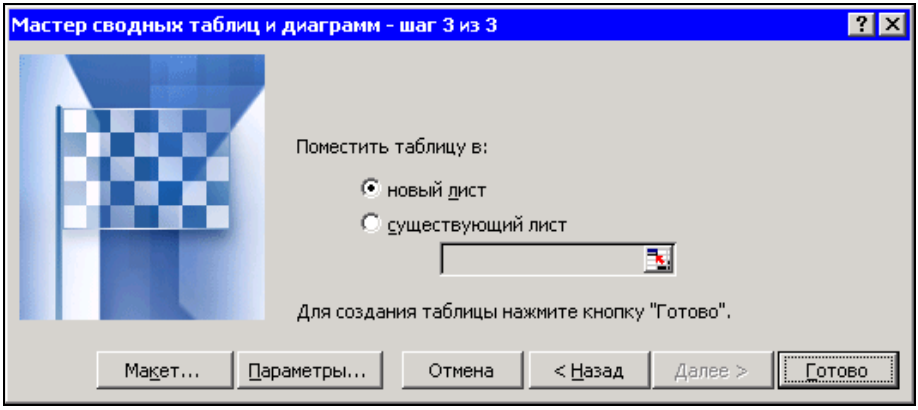


Рис. 19.11. Третье окно мастера сводных таблиц

4. Первоначально, для создания структуры сводной таблицы, нажмите кнопку **Макет**. На экране отобразится окно **Макет** мастера сводных таблиц (рис. 19.12). Прежде чем создавать макет сводной таблицы, надо определиться, какая информация будет вводиться в области строк, столбцов, данных и страниц сводной таблицы.

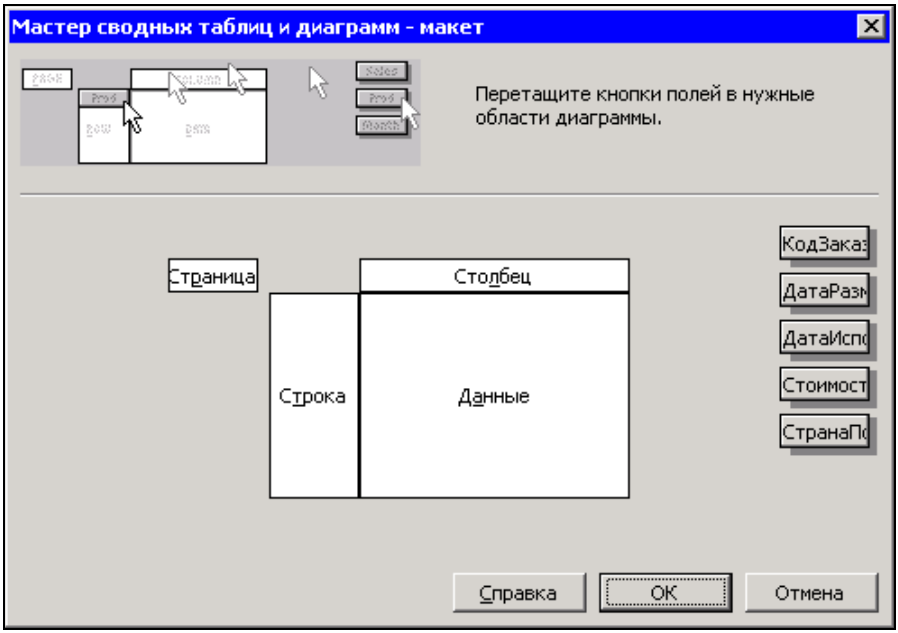


Рис. 19.12. Окно **Макет** мастера сводных таблиц

- Итак, для создания структуры сводной таблицы выберите поле, содержащее данные, по которым нужно подвести итоги, и перетащите соответствующую кнопку в область **Данные**. В нашем случае будем подводить итоги по полю **Стоимость**. Для того чтобы выбрать операцию, по которой подводятся итоги, дважды щелкните по полю **Стоимость**, расположенному в области **Данные.** На экране отобразится окно **Вычисление поля сводной таблицы** (рис. 19.13). В списке **Операция** перечислены допустимые операции: `Сумма`, `Количество`, `Среднее`, `Максимум`, `Минимум`, `Произведение`, `Количество чисел`, `Смещенное отклонение`, `Несмещенное отклонение`, `Смещенная дисперсия`, `Несмещенная дисперсия`. В нашем случае выберите `Сумма`. Список **Дополнительные вычисления** позволяет расширить множество допустимых операций. Его допустимые значения: `Нет`, `Отличие`, `Доля`, `Приведенное отличие`, `С нарастающим итогом в поле`, `Доля от суммы по строке`, `Доля от суммы по столбцу`, `Доля от общей суммы`, `Индекс`. Список **Дополнительные вычисления** отображается и скрывается последовательным нажатием кнопки **Дополнительно >>**. В нашем случае выберите в списке `Нет` и нажмите кнопку **OK**. Мы вернемся в окно **Макет**.
- Для того чтобы поместить элементы поля в строках с меткой в левой части таблицы, перетащите кнопку для выбранного поля в область **Строка**. В нашем случае по строкам будут размещаться страны, поэтому перетащите поле **СтранаПолучателя** в область **Строка**.

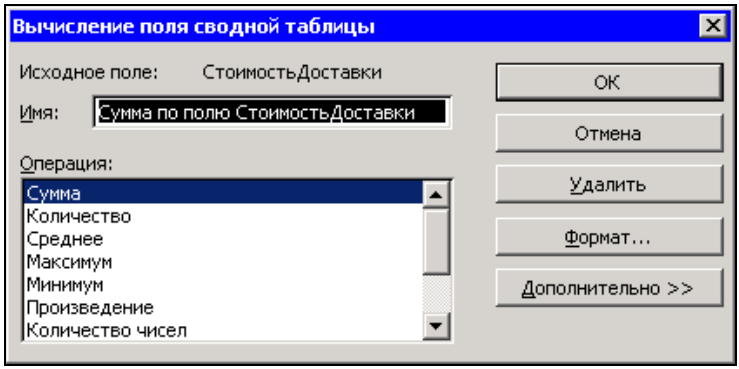


Рис. 19.13. Окно **Вычисление поля сводной таблицы**

- Для того чтобы поместить элементы поля в столбцах с меткой в верхней части таблицы, перетащите кнопку для выбранного поля в область **Столбец**. В нашем случае перетаскивать ничего не надо.
- Для того чтобы поместить элементы поля по страницам, что позволит работать со сводной таблицей как со стопкой листов, перетащите кнопку для выбранного поля в область **Страница**. В нашем случае перетаскивать ничего не надо.

Итак, в окне **Макет** мастера сводных таблиц создана структура будущей сводной таблицы (рис. 19.14). Нажмите на кнопку **ОК**, и мы вернемся к третьему окну мастера сводных таблиц (см. рис. 19.11).

Примечание

Теперь можно позаботиться о параметрах сводной таблицы. Как правило, те значения параметров, которые установлены по умолчанию, являются оптимальными, и их не надо изменять. Но для того чтобы лучше разобраться с тем, как настраивается сводная таблица, нажмите кнопку **Параметры**. На экране отобразится окно **Параметры сводной таблицы** (рис. 19.15). В окне под полем **Имя,** в которое вводится имя сводной таблицы, находятся две группы параметров: **Формат** и **Данные**. Из группы **Формат** отметим только флажки **общая сумма по строкам** и **общая сумма по столбцам**, которые определяют, надо ли подводить итоги по строкам и столбцам. При построении сводной таблицы все данные копируются в скрытую *кэш-память*. Флажок **сохранить данные вместе с таблицей** определяет, будут ли в кэш-памяти при изменениях сводной таблицы сохраняться старые сведения или обновленные.

5. В группе **Поместить таблицу в** третьего окна мастера сводных таблиц (см. рис. 19.11) имеются два переключателя **новый лист** и **существующий лист**, которые задают сводной таблице местоположение. Если выбран переключатель **существующий лист**, то в поле надо привести ссылку на левую верхнюю ячейку диапазона, где будет располагаться таблица. В нашем случае выберите переключатель **новый лист** и нажмите кнопку **Готово**. Сводная таблица построена (рис. 19.16).

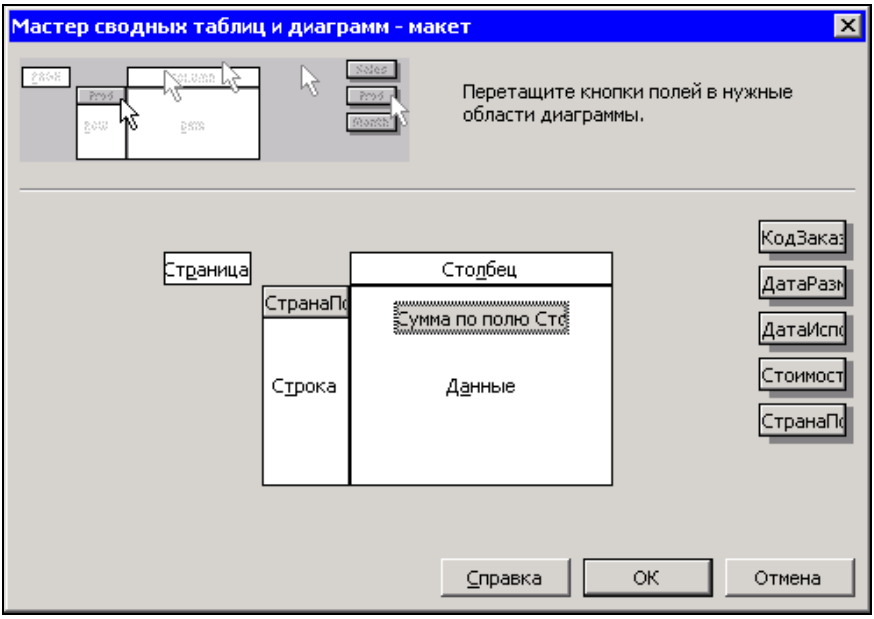


Рис. 19.14. Окно **Макет** со сконфигурированным макетом сводной таблицы

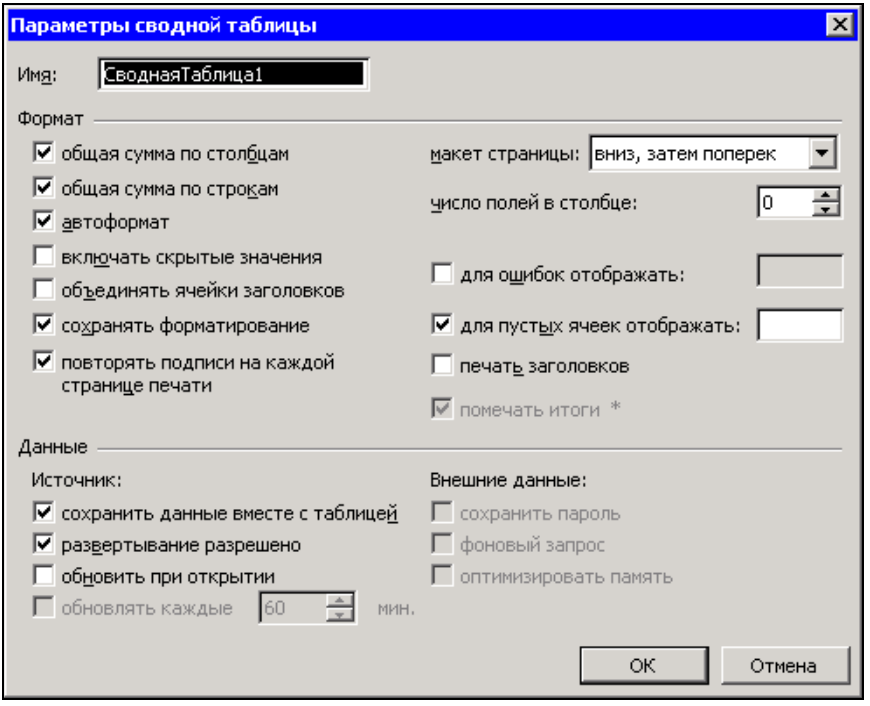


Рис. 19.15. Окно **Параметры сводной таблицы**

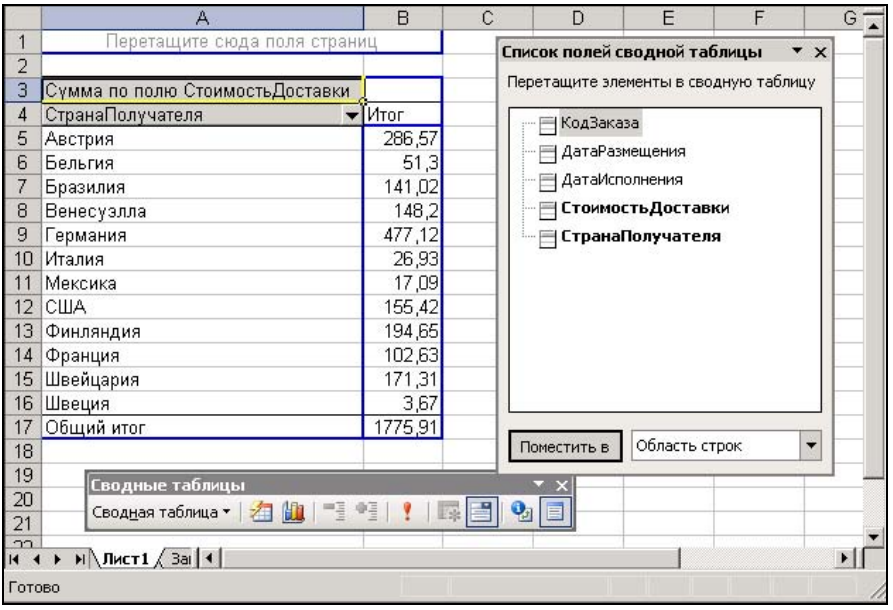


Рис. 19.16. Сводная таблица с панелью инструментов **Сводные таблицы**

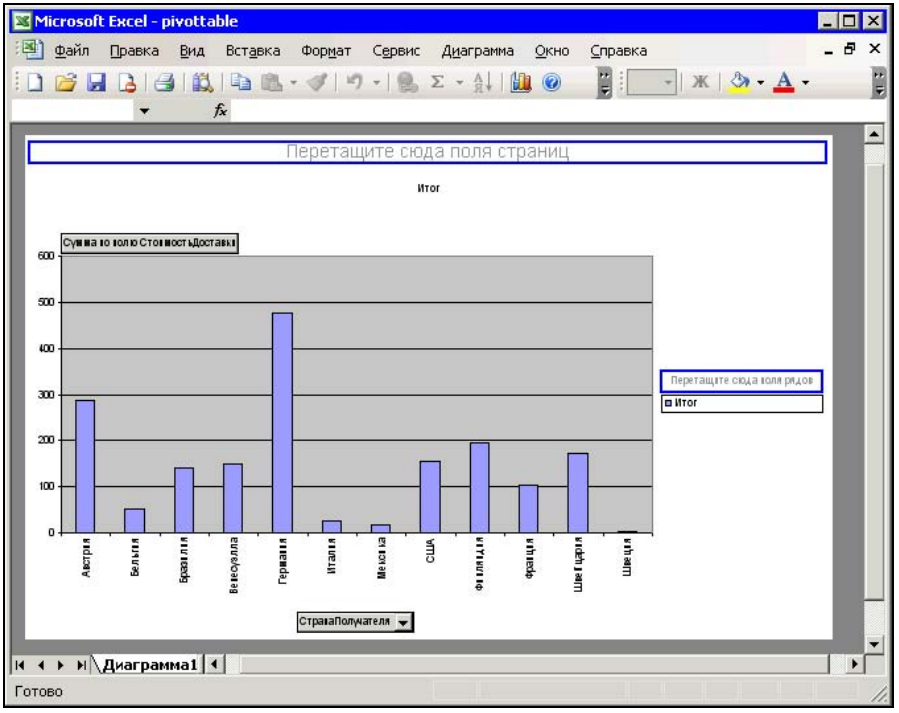


Рис. 19.17. Сводная диаграмма

Щелчок на кнопке в поле **СтранаПолучателя** приводит к отображению списка с именами получателей. Этот список позволяет отображать информацию в сводной таблице, как обо всех получателях, так и только о тех, которые выбраны.

Щелчок на кнопке **Мастер диаграмм** панели инструментов **Сводные таблицы** приводит к построению сводной диаграммы на базе сводной таблицы (рис. 19.17).

Обновление данных

Сводная таблица отображает содержание кэш-памяти, а не исходного диапазона данных. Если уже после создания сводной таблицы вы измените данные, на основе которых она была построена, то это не приведет к автоматическому изменению данных в сводной таблице. Сводная таблица не является динамической таблицей, автоматически перестраиваемой при обновлении данных, на основе которых она построена. Для обновления сводной таблицы нужно выделить любую ячейку сводной таблицы и выбрать команду **Данные | Обновить данные** или нажать кнопку панели инструментов **Сводные таблицы**. При этом в кэш-память будут вновь занесены сведения, на основе которых строится сводная таблица.

Удаление сводной таблицы

Для удаления сводной таблицы надо выбрать какую-нибудь ее ячейку. Затем в панели инструментов **Сводные таблицы** нажать кнопку **Сводная таблица** и в появившемся меню выбрать команду **Выделить | Таблицу целиком**, после чего выбрать команду **Правка | Очистить | Все**.

Объекты, связанные со сводной таблицей

Со сводной таблицей связан ряд объектов, которые перечислены в табл. 19.4.

Таблица 19.4. *Объекты, связанные со сводной таблицей*

Объект	Описание
`PivotTable`	Сводная таблица
`PivotCache`	Кэш-память, отведенная под сводную таблицу. Этот объект возвращается методом `PivotCache` объекта `PivotTable`
`PivotCell`	Ячейка сводной таблицы. Данный объект можно получить при помощи свойства `PivotCell` объекта `Range`
`PivotField`	Поле сводной таблицы. Этот объект возвращается методом `PivotFields` объекта `PivotTable`

Таблица 19.4 *(окончание)*

Объект	Описание
PivotFormula	Формула, по которой подводится итог в сводной таблице. Этот объект возвращается методом PivotFormulas объекта PivotTable
PivotItem	Элемент поля сводной таблицы. Этот объект возвращается методом PivotItems объекта PivotTable
PivotLayout	Данные о размещении полей и осей сводной таблицы и сводной диаграммы. Этот объект можно получить при помощи свойства PivotLayout объекта Chart

Все эти объекты являются членами соответствующих семейств, а именно, PivotTables, PivotFields, PivotFormulas, PivotItems и PivotItemList.

Метод *PivotTableWizard*

Программно сводная таблица создается методом PivotTableWizard объектов Worksheet, Workbook или PivotTable. Вручную на рабочем листе сводная таблица конструируется при помощи мастера сводной таблицы, активизируемого командой **Данные | Сводная таблица**.

```
PivotTableWizard(SourceType, SourceData, TableDestination,
↳TableName, RowGrand, ColumnGrand, SaveData, HasAutoFormat, AutoPage,
↳Reserved, BackgroundQuery, OptimizeCache, PageFieldOrder,
↳PageFieldWrapCount, ReadData, Connection)
```

Все параметры этого метода являются необязательными:

- *SourceType* — тип источника данных. Допустимые значения: XlConsolidation (консолидация нескольких диапазонов рабочих листов), xlDatabase (список или база данных MS Excel), xlExternal (внешняя база данных), xlPivotTable (сводная таблица);
- *SourceData* — определяет вид источника данных в зависимости от значения параметра *SourceType*. Если параметр *SourceType* принимает значение xlDatabase, то источником данных является диапазон. Если xlExternal, то массив строк, содержащий строку связи ODBC и SQL-оператор. Если xlConsolidation, то массив диапазонов. Если xlPivotTable, то имя существующей сводной таблицы;
- *TableDestination* — диапазон, где будет размещена сводная таблица;
- *TableName* — имя создаваемой сводной таблицы;
- *RowGrand* — параметр, принимающий логические значения. Если его значение равно True, то отображается суммарный итог по строкам сводной таблицы. Если его значение равно False, то итог не отображается;

- *ColumnGrand* — параметр, принимающий логические значения. Если его значение равно True, то отображается суммарный итог по столбцам сводной таблицы. Если его значение равно False, то итог не отображается;
- *SaveData* — параметр, принимающий логические значения. Если его значение равно True, то данные сохраняются вместе со сводной таблицей. Если его значение равно False, то сохраняется только сводная таблица;
- *HasAutoFormat* — параметр, принимающий логические значения. Если его значение равно True, то происходит автоматическое переформатирование сводной таблицы при изменении данных;
- *AutoPage* — применяется только при значении параметра SourceType равном xlConsolidation. Допустимые значения True (MS Excel создает поле страницы) и False (пользователь должен создать поле);
- Reserved — не используется;
- BackgroundQuery — параметр, принимающий логические значения. Если его значение равно True, то Excel выполняет запрос в фоновом режиме, если False — в последовательном;
- OptimizeCache — параметр, принимающий логические значения. Если его значение равно True, то сводная таблица создается в режиме оптимизации, что применяется для сводных таблиц, обрабатывающих большие базы данных. Если его значение равно False, то оптимизация выключена, что убыстряет создание сводной таблицы;
- PageFieldOrder — задает последовательность, в которой поля добавляются в таблицу. Допустимые значения: xlDownThenOver и xlOverThenDown;
- PageFieldWrapCount — задает номер поля, с которого начинается новая страница. По умолчанию 0, т. е. отменена разбивка на страницы;
- ReadData — параметр, принимающий логические значения. Если его значение равно True, то данные сразу считываются в кэш. Если его значение равно False, то данные считываются в кэш по мере необходимости;
- Connection — используется для указания источника данных ODBC, источника данных URL и имени файла, содержащего запрос.

С методом PivotTableWizard тесно связано свойство PivotTables объекта Worksheet, которое возвращает семейство сводных таблиц рабочего листа.

Объект *PivotTable*

Объект PivotTable инкапуслирует в себе данные о сводной таблице. Этот объект является членом семейства PivotTables. В табл. 19.5 приведены методы объекта PivotTable, а в табл. 19.6 — наиболее часто используемые свойства этого объекта.

Таблица 19.5. *Методы объекта* `PivotTable`

Метод	Описание
`AddDataField`	Добавляет поля с данными
`AddFields`	Добавляет строки, столбцы и страницы в сводную таблицу
`CalculatedFields`	Возвращает семейство `CalculatedFields` всех вычисляемых полей сводной таблицы
`Format`	Задает формат отчета сводной таблицы
`GetData`	Возвращает данные из указанной ячейки сводной таблицы
`GetPivotData`	Возвращает диапазон с информацией о сводной таблице
`ListFormulas`	Создает список формул на отдельном листе
`PivotCache`	Возвращает объект `PivotCache`
`PivotFields`	Возвращает семейство `PivotFields`
`PivotSelect`	Выбирает часть сводной таблицы
`PivotTableWizard`	Конструирует сводную таблицу по данной таблице
`RefreshTable`	Обновляет сводную таблицу. Для перерасчета сводной таблицы вручную надо ее выделить и выбрать команду **Данные | Обновить данные**
`ShowPages`	Устанавливает содержимое области **Страница**
`Update`	Обновляет связи в сводной таблице

Таблица 19.6. *Свойства объекта* `PivotTable`

Свойство	Описание
`ColumnFields`, `RowFields`, `DataFields`, `PageFields`	Возвращают объект (либо единичное поле, либо семейство полей), который является столбцом (строкой, данными и страницей) сводной таблицы
`VisibleFields`, `HiddenFields`	Возвращают объект, являющийся либо единичным полем, либо семейством полей, который в данный момент отображается (скрыт) в сводной таблице

Объект *PivotCache*

Объект `PivotCache` представляет собой кэш-память, выделенную под конкретную сводную таблицу. Этот объект является членом семейства `PivotCaches` и возвращается методом `PivotCache` объекта `PivotTable`.

Основным методом семейства PivotCaches является метод Add. Он имеет следующий синтаксис:

Add(*SourceType*, *SourceData*), где:

- *SourceType* — обязательный параметр, задающий тип данных, на основе которых строится сводная таблица. Допустимыми значениями являются следующие константы XlPivotTableSourceType: XlConsolidation (консолидация нескольких диапазонов рабочих листов), xlDatabase (список или база данных Excel), xlExternal (внешняя база данных), xlPivotTable (сводная таблица);
- *SourceData* — необязательный параметр, определяющий вид источника данных в зависимости от значения параметра *SourceType*. Этот параметр является обязательным, если значение параметра *SourceType* отлично от xlExternal.

Объект PivotCache имеет ряд методов, перечисленных в табл. 19.7.

Таблица 19.7. *Методы объекта* PivotCache

Метод	Описание
CreatePivotTable	Создает PivotTable объект. CreatePivotTable(*TableDestination*, *TableName*, *ReadData*), где: *TableDestination* — необязательный параметр, дающий ссылку на ячейку, в которой будет располагаться верхний левый угол сводной таблицы. Если сводная таблица создается на новом рабочем листе, то значение этого параметра должно быть равно пустой строке; *TableName* — необязательный параметр, задающий имя сводной таблицы; *ReadData* — необязательный параметр, принимающий логические значения и определяющий, надо ли сводную таблицу строить по всей внешней базе данных
MakeConnection	Устанавливает соединения с кэш-памятью
Refresh	Обновляет кэш-память
ResetTimer	Переустанавливает таймер обновления кэш-памяти
SaveAsODC	Запись кэш-памяти в файл ODC (Microsoft Office Data Connection)

Объект *PivotField*

Объект PivotField представляет собой поле сводной таблицы. Этот объект является членом семейства PivotFields и возвращается методом PivotField объекта PivotTable. В табл. 19.8 перечислены методы, а в табл. 19.9 — основные свойства объекта PivotField.

Таблица 19.8. *Методы объекта* `PivotField`

Свойство	Описание
`AddPageItem`	Добавляет элемент в поле
`AutoShow`	Задает правило автоматического отображения элементов полей
`AutoSort`	Задает правило автоматической сортировки полей
`CalculatedItems`	Возвращает семейство `CalculatedItems`
`Delete`	Удаляет поле
`PivotItems`	Возвращает семейство `PivotItems` всех элементов поля

Таблица 19.9. *Свойства объекта* `PivotField`

Свойство	Описание
`Orientation`	Возвращает местоположение поля в сводной таблице. Допустимые значения: `xlColumnField`, `xlDataField`, `xlHidden`, `xlPageField` и `xlRowField`
`Name`	Имя поля
`Function`	Функция, по которой в поле подводится итог. Допустимые значения: `xlAverage`, `xlCount`, `xlCountNums`, `xlMax`, `xlMin`, `xlProduct`, `xlStDev`, `xlStDevP`, `xlSum`, `xlVar` и `xlVarP`
`Position`	Возвращает позицию поля (первое, второе и т. д.) среди полей того же местоположения

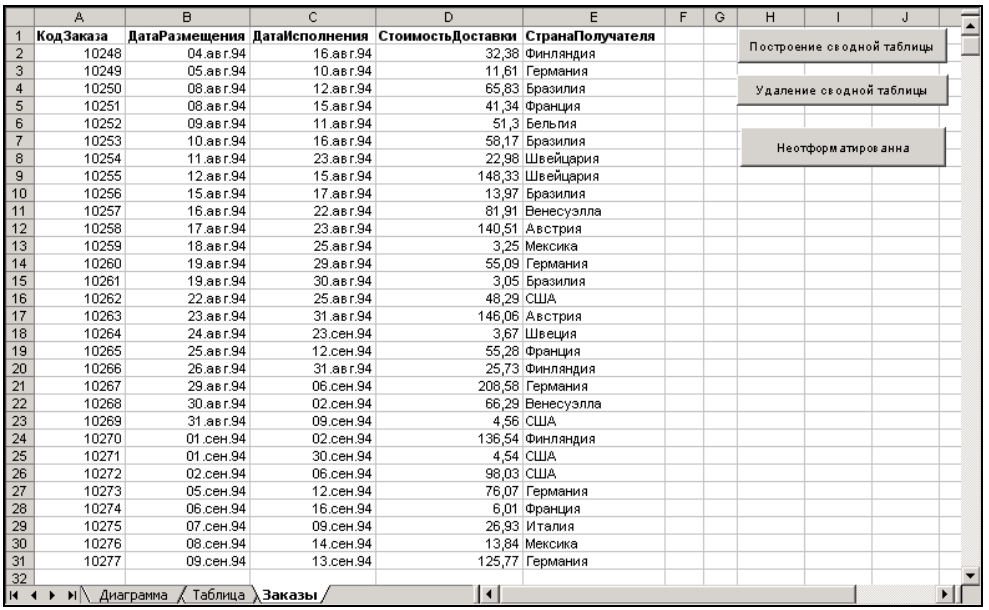
Пример приложения, помогающего построить и обновлять сводную таблицу

Приведем простое приложение, которое автоматизирует процесс построения, обновления и удаления сводной таблицы и сводной диаграммы. Автоматизация построения сводной таблицы в ситуации, когда список заказов постоянно обновляется, бесспорно, является хорошим подспорьем.

Итак, в проекте имеются три листа (рис. 19.18):

- рабочий лист **Заказы** — на этом листе расположена таблица, на базе которой будут строиться сводная таблица и диаграмма. Данная таблица состоит из пяти полей: **КодЗаказа, ДатаРазмещения, ДатаИсполнения, СтоимостьДоставки** и **СтранаПолучателя**. Заголовки полей расположены в ячейках **A1**, **B1**, **C1**, **D1**, **E1**. Число полей произвольно и может изменяться со временем;
- рабочий лист **Таблица** — на нем создается сводная таблица. Этот лист необязательный. Он автоматически создается приложением;

- лист диаграмм **Диаграмма** — на нем конструируется сводная диаграмма. Этот лист необязательный. В случае его отсутствия он автоматически будет создан при построении сводной диаграммы.



	КодЗаказа	ДатаРазмещения	ДатаИсполнения	СтоимостьДоставки	СтранаПолучателя
2	10248	04.авг.94	16.авг.94	32,38	Финляндия
3	10249	05.авг.94	10.авг.94	11,61	Германия
4	10250	08.авг.94	12.авг.94	65,83	Бразилия
5	10251	08.авг.94	15.авг.94	41,34	Франция
6	10252	09.авг.94	11.авг.94	51,3	Бельгия
7	10253	10.авг.94	16.авг.94	58,17	Бразилия
8	10254	11.авг.94	23.авг.94	22,98	Швейцария
9	10255	12.авг.94	15.авг.94	148,33	Швейцария
10	10256	15.авг.94	17.авг.94	13,97	Бразилия
11	10257	16.авг.94	22.авг.94	81,91	Венесуэлла
12	10258	17.авг.94	23.авг.94	140,51	Австрия
13	10259	18.авг.94	25.авг.94	3,25	Мексика
14	10260	19.авг.94	29.авг.94	55,09	Германия
15	10261	19.авг.94	30.авг.94	3,05	Бразилия
16	10262	22.авг.94	25.авг.94	48,29	США
17	10263	23.авг.94	31.авг.94	146,06	Австрия
18	10264	24.авг.94	23.сен.94	3,67	Швеция
19	10265	25.авг.94	12.сен.94	55,28	Франция
20	10266	26.авг.94	31.авг.94	25,73	Финляндия
21	10267	29.авг.94	06.сен.94	208,58	Германия
22	10268	30.авг.94	02.сен.94	66,29	Венесуэлла
23	10269	31.авг.94	09.сен.94	4,56	США
24	10270	01.сен.94	02.сен.94	136,54	Финляндия
25	10271	01.сен.94	30.сен.94	4,54	США
26	10272	02.сен.94	06.сен.94	98,03	США
27	10273	05.сен.94	12.сен.94	76,07	Германия
28	10274	06.сен.94	16.сен.94	6,01	Франция
29	10275	07.сен.94	09.сен.94	26,93	Италия
30	10276	08.сен.94	14.сен.94	13,84	Мексика
31	10277	09.сен.94	13.сен.94	125,77	Германия

Рис. 19.18. Построение и обновление сводной таблицы

На рабочем листе **Заказы** имеются три кнопки:

- кнопка **Построение сводной таблицы** конструирует сводную таблицу и диаграмму, причем, автоматически определяется число записей в таблице на листе **Заказы**;
- кнопка **Удаление сводной таблицы** удаляет листы со сводной таблицей и диаграммой;
- Кнопка **Неотформатированно** изменяет формат отображения сводной таблицы, причем при изменении формата изменяется надпись, отображаемая на кнопке.

Для реализации данного проекта в окне **Properties** установите значения свойств кнопок, как показано в табл. 19.10 и в модуле рабочего листа наберите следующий код (листинг 19.5).

Таблица 19.10. *Значения свойств элементов управления, установленные в окне* ***Properties***

Элемент управления	Свойство	Значение
Кнопка	`Name`	`cmdPTBuilder`
	`Caption`	`Построение сводной таблицы`

Таблица 19.10 *(окончание)*

Элемент управления	Свойство	Значение
Кнопка	Name	CmdPTDelete
	Caption	Удаление сводной таблицы
Кнопка	Name	cmdTypeReport
	Caption	Неотформатированно

Листинг 19.5. Построение и обновление сводной таблицы. Модуль рабочего листа

```
Const sheetName As String = "Заказы"
Const PTName As String = "ReportPivotTable"
Const PTsheetName As String = "Таблица"
Const PTChartName As String = "Диаграмма"

Private Sub cmdPTBuilder_Click()
   Application.ScreenUpdating = False
   PTBuilder
   Application.ScreenUpdating = True
End Sub

Private Sub cmdPTDelete_Click()
   Application.ScreenUpdating = False
   Application.DisplayAlerts = False
   DeleteSheet PTsheetName
   DeleteSheet PTChartName
   Application.DisplayAlerts = False
   Application.ScreenUpdating = True
End Sub

Sub PTBuilder()
   cmdPTDelete_Click
   Dim r As Range
   Set r = Worksheets(sheetName).Range("A1").CurrentRegion
   Dim adrs As String
   adrs = r.Address(ReferenceStyle:=xlR1C1)
   adrs = sheetName & "!" & adrs
   ActiveWorkbook.PivotCaches.Add(SourceType:=xlDatabase, _
```

```
                                    SourceData:=adrs) _
        .CreatePivotTable TableDestination:="", _
                          TableName:=PTName, _
                          DefaultVersion:=xlPivotTableVersion10
    ActiveSheet.Name = PTsheetName
    Sheets(PTsheetName).PivotTableWizard _
                        TableDestination:=ActiveSheet.Range("A1")
    Sheets(PTsheetName).PivotTables(PTName).AddFields _
                        RowFields:="СтранаПолучателя"
    Sheets(PTsheetName).PivotTables(PTName). _
              PivotFields("СтоимостьДоставки").Orientation = xlDataField
    Charts.Add
    ActiveChart.SetSourceData _
                    Source:=Sheets(PTsheetName).Range("A1").CurrentRegion
    ActiveChart.Location Where:=xlLocationAsNewSheet
    ActiveSheet.Name = PTChartName
    Sheets(sheetName).Select
    ActiveWorkbook.ShowPivotTableFieldList = False
End Sub

Sub DeleteSheet(s As String)
    On Error Resume Next
        Sheets(s).Delete
    On Error GoTo 0
End Sub

Private Sub cmdTypeReport_Click()
    Static frm As Integer
    On Error GoTo errorhandler
    Select Case frm
       Case 0
          Sheets(PTsheetName).PivotTables(PTName).Format xlTable1
          cmdTypeReport.Caption = "Формат Table1"
       Case 1
          Sheets(PTsheetName).PivotTables(PTName).Format xlPTClassic
          cmdTypeReport.Caption = "Формат Classic"
       Case 2
          Sheets(PTsheetName).PivotTables(PTName).Format xlReport1
```

```
            cmdTypeReport.Caption = "Формат Report1"
        Case 3
            Sheets(PTsheetName).PivotTables(PTName).Format xlPTNone
            cmdTypeReport.Caption = "Неотформатированна"
    End Select
    frm = (frm + 1) Mod 4
errorhandler:
End Sub
```

Списки

MS Excel предоставляет мощное средство **Список** для просмотра, редактирования и обновления списка данных. Список создается командой **Данные | Список | Создать список.** Может быть создан как пустой список, так и список на основе указанного диапазона данных. На рабочем листе может располагаться несколько списков. Команда **Данные | Список | Преобразовать в диапазон** преобразует список в обычный диапазон. Команда **Данные | Список | Опубликовать список** публикует список, а команда **Данные | Список | Просмотреть список на сервере** реализует предварительный просмотр списка на сервере (Microsoft Windows SharePoint Services). Опубликованные данные, содержащиеся в списке, могут быть доступны для совместного использования другим уполномоченным пользователями. Команда **Данные | Список | Изменить размер списка** изменяет размеры списка. А команда **Данные | Список | Строка итогов** добавляет в список строки с подведением итогов по строкам.

Используя средства фильтрации данных, встроенные в список можно сортировать данные как в убывающем, так и возрастающем порядке. Допустимо также создание пользовательского способа упорядочивания. Возможна фильтрация данных в соответствии с установленным пользователем критерием. Для того чтобы добавить строку в список, достаточно ввести данные в строку, помеченную звездочкой. Для удаления строки из списка — удалить ее командой **Правка | Удалить | Строку**.

Программно список реализуется объектом `ListObject`, а все списки рабочего листа — в семействе `ListObjects`. Кроме того, имеются объекты `ListRow` и `ListColumn` и семейства `ListRows` и `ListColumns`, предоставляющие средства управления списком.

В качестве примера создадим следующее приложение. На рабочем листе имеется таблица данных с полями **КодЗаказа**, **ДатаРазмещения**, **СтоимостьДоставки, СтранаПолучателя**. Кроме того, разместите на листе две кнопки, флажок и список (рис. 19.19).

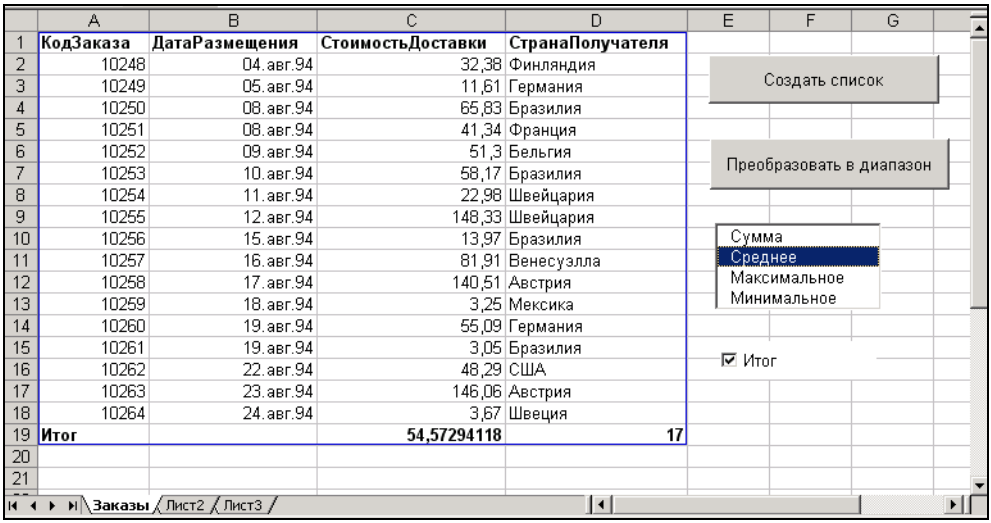


Рис. 19.19. Работа со списком

В окне **Properties** установите значения свойств элементов управления, как показано в табл. 19.11 и в модуле рабочего листа наберите код из листингов 19.6, *а*, *б*.

Таблица 19.11. *Значения свойств элементов управления, установленные в окне* ***Properties***

Элемент управления	Свойство	Значение
Кнопка	Name	cmdCreateList
	Caption	Создать список
Кнопка	Name	cmdToRange
	Caption	Преобразовать в диапазон
Флажок	Name	chkTotal
	Caption	Итог
Список	Name	lstOperation

Кнопки **Создать список** и **Преобразовать в диапазон** конструируют и удаляют список. Флажок **Итог** управляет отображением итогов, а список задает тип формулы, по которой подводится итог.

Листинг 19.6, *а*. Модуль ЭтаКнига

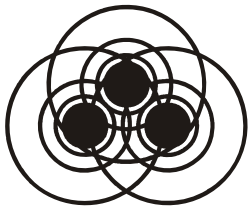
```
Private Sub Workbook_Open()
   On Error Resume Next
   With Worksheets(1).lstOperation
     .AddItem "Сумма"
     .AddItem "Среднее"
     .AddItem "Максимальное"
     .AddItem "Минимальное"
   End With
End Sub
```

Листинг 19.6, *б*. Модуль рабочего листа

```
Const listName As String = "MyList"

Private Sub cmdCreateList_Click()
   On Error Resume Next
   Dim r As Range
   Set r = Range("A1").CurrentRegion
   ActiveSheet.ListObjects.Add(xlSrcRange, r, , xlYes).Name = listName
   chkTotal.Value = False
   lstOperation.ListIndex = 0
End Sub

Private Sub cmdToRange_Click()
   On Error Resume Next
   ActiveSheet.ListObjects(listName).ShowTotals = False
   ActiveSheet.ListObjects(listName).Unlist
End Sub

Private Sub chkTotal_Click()
   On Error Resume Next
   ActiveSheet.ListObjects(listName).ShowTotals = chkTotal.Value
End Sub

Private Sub lstOperation_Change()
   On Error Resume Next
```

```
   Select Case lstOperation.ListIndex
      Case 0
         ActiveSheet.ListObjects(listName).ListColumns( _
               "СтоимостьДоставки").TotalsCalculation = _
               xlTotalsCalculationSum
      Case 1
          ActiveSheet.ListObjects(listName).ListColumns( _
              "СтоимостьДоставки").TotalsCalculation = _
               xlTotalsCalculationAverage
      Case 2
          ActiveSheet.ListObjects(listName).ListColumns( _
              "СтоимостьДоставки").TotalsCalculation = _
               xlTotalsCalculationMin
      Case 3
            ActiveSheet.ListObjects(listName).ListColumns( _
               "СтоимостьДоставки").TotalsCalculation = _
               xlTotalsCalculationMax
   End Select
End Sub
```

Глава 20



VBA и Web

Отсылка сообщений по электронной почте

Для отсылки рабочей книги по электронной почте надо воспользоваться методом `SendMail` объекта `Workbook`. У этого метода имеются три параметра, причем только первый из них является обязательным. Первый задает адрес, по которому отсылается почта, второй — ее тему, а третий задает, надо ли производить отправку почты с подтверждением. Следующий код (листинг 20.1) демонстрирует, как можно использовать метод `SendMail` при отсылке активной рабочей книги.

Листинг 20.1. Отсылка по электронной почте рабочей книги

```
Sub SendBook(wb As Workbook, address As String, subject As String)
   wb.SendMail address, subject
End Sub

Sub DemoSendBook()
   SendBook ActiveWorkbook, "someone@somewhere.ru", "test message"
End Sub
```

Можно отослать не целую книгу, а только один лист. Для этого достаточно воспользоваться следующим кодом (листинг 20.2), который на базе специфицированного листа создает временную рабочую книгу, состоящую из этого листа, затем ее отсылает, после чего уничтожает.

Листинг 20.2. Отсылка по электронной почте рабочего листа

```
Sub SendSheet(ws As Worksheet, address As String, subject As String)
   ActiveSheet.Copy
   ActiveWorkbook.SaveAs "Part of " & _
```

```
            ThisWorkbook.Name & " " & Format(Now, "h-mm-ss"), _
             xlWorkbookNormal
   ActiveWorkbook.SendMail address, subject
   ActiveWorkbook.ChangeFileAccess xlReadOnly
   Kill ActiveWorkbook.FullName
   ActiveWorkbook.Close False
End Sub

Sub DemoSendSheet()
   SendSheet ActiveSheet, "someone@somewhere.ru", "test message"
End Sub
```

Задание параметров Web-страницы

Прежде чем сохранять документ как Web-страницу, разумно установить ее параметры, которые инкапсулируются в объекте `DefaultWebOptions`. Например, следующий код (листинг 20.3) с помощью свойства `PixelsPerInch` устанавливает число пикселей, приходящихся на один дюйм и позволяет сохранять графические объекты в VML формате (Vector Markup Language). Это векторный формат, он сохраняет графические файлы в файлах меньших размерах. При этом надо иметь в виду, что Microsoft Internet Explorer, начиная с версии 5.0, его поддерживает, но другие браузеры его не поддерживают. Свойство `Encoding` задает кодировку страницы. Свойство `AllowPNG` позволяет использовать PNG формат (Portable Network Graphics) для графики, более экономный, чем стандартные графические форматы.

Листинг 20.3. Задание параметров Web-страницы

```
With Application.DefaultWebOptions
   .RelyOnVML = True
   .AllowPNG = True
   .PixelsPerInch = 96
   .Encoding = msoEncodingCyrillic
End With
```

Предварительный просмотр документа как Web-страницы

Предварительный просмотр документа как Web-страницы реализуется методом `WebPagePreview`.

```
ActiveWorkbook.WebPagePreview
```

Сохранение документа как Web-страницы

Сохранить документ как Web-страницу можно методом `SaveAs`, указав в качестве значения его параметра `FileFormat` константу `xlHTML`. Например, в следующем коде активная рабочая книга сохраняется как Web-страница Page.htm.

```
ActiveWorkbook.SaveAs Filename:="C:\Page.htm", FileFormat:=xlHtml
```

Объект *PublishObject*

Объект `PublishObject` представляет собой элемент документа, сохраненный как Web-страница. Такой объект может автоматически обновлять свои данные при соответствующих изменениях в оригинальном сохраненном документе, если значение его свойства `AutoRepublish` установлено равным `True`. Все объекты `PublishObject` образуют семейство `PublishObjects`. Новый элемент в семейство добавляется методом `Add`.

`Add(`*`SourceType`*`, `*`FileName`*`, `*`Sheet`*`, `*`Source`*`, `*`HtmlType`*`, `*`DivID`*`, `*`Title`*`)`, где:

- *`SourceType`* — специфицирует источник данных. Допустимыми значениями могут быть следующие `XlSourceType` константы: `xlSourceAutoFilter`, `xlSourceChart`, `xlSourcePivotTable`, `xlSourcePrintArea`, `xlSourceQuery`, `xlSourceRange`, `xlSourceSheet`, `xlSourceWorkbook`;
- *`FileName`* — имя файла (или его URL при сетевом расположении документа);
- *`Sheet`* — имя листа;
- *`Source`* — имя источника данных;
- *`HtmlType`* — специфицирует, сохранен ли объект как итеративная компонента, либо как статический рисунок или текст. Допустимыми значениями могут быть следующие `XlHtmlType` константы: `xlHtmlCalc` (компонента рабочего листа), `xlHtmlChart` (компонента диаграммы), `xlHtmlList` (компонента сводной таблицы), `xlHtmlStatic` (статический элемент, только для просмотра в окне браузера);
- *`DivID`* — значение идентификатора для `DIV` тега, однозначно специфицирующего компоненту;
- *`Title`* — устанавливает заголовок Web-страницы.

В следующем примере значения из диапазона **A1:B2** рабочего листа **Отчет** активной книги добавляются в страницу test.htm, расположенную по адресу **http:\localhost**, в виде компоненты рабочего листа.

```
Sub PublishRange()
   Dim pb As PublishObject
```

```
   Set pb = ActiveWorkbook.PublishObjects.Add( _
      SourceType:=xlSourceRange, _
       Filename:="http:\localhost\test.htm", _
       Sheet:="Отчет", _
       Source:="A1:B2", _
       HtmlType:=xlHtmlCalc, _
       DivID:="RgnA1B2", _
       Title:="Тестовая страница")
    pb.AutoRrepublish = True
    pb.Publish
End Sub
```

Для того чтобы пользователь мог просматривать сохраненные итеративные Web-страницы, на его компьютере должны быть установлены компоненты Microsoft Office Web Components, лицензия на использование Microsoft Office и Microsoft Internet Explorer, начиная с версии 4.01. Для просмотра неитеративных страниц достаточно иметь установленный Microsoft Internet Explorer, начиная с версии 4.01.

Гиперссылки

Гиперссылки предоставляют средство для путешествия между документами, расположенными как на вашем компьютере, так и в сети. Информация о гиперссылке инкапсулируется в объекте `Hyperlink`, который в своей совокупности образует семейство `Hyperlinks`. Следующий простой код (листинг 20.4) демонстрирует, как на активном листе можно найти все гиперссылки и окрасить ячейки, содержащие их желтый цвет.

Листинг 20.4. Окраска всех гиперссылок рабочего листа

```
Sub ShowHyperlink()
   Dim r As Range
   Dim lnk As Hyperlink
   Dim IsFirstCell As Boolean
   If ActiveSheet.Hyperlinks.Count > 0 Then
      IsFirstCell = True
      For Each lnk In ActiveSheet.Hyperlinks
         If IsFirstCell Then
            Set r = lnk.Range
            IsFirstCell = False
         Else
```

```
            Set r = Application.Union(r, lnk.Range)
         End If
      Next
     r.Interior.Color = vbYellow
   End If
End Sub
```

Отсылка сообщения с помощью гиперссылки

Используя гиперссылки можно не только путешествовать по сети, но и отсылать электронную почту. Для этого достаточно временно создать гиперссылку типа `mailto:`, а после отправки сообщения, удалить ее. В коде из листинга 20.5 содержимое ячейки **A1** отправляется в качестве сообщения по адресу **somebody@somewhere.com**.

Листинг 20.5. Отсылка сообщения с помощью гиперссылки

```
Sub SendMailByHyperlink()
   Dim EMailAddress As String
   Dim EMailSubject As String
   Dim EMailMessage As Variant
   EMailAddress = "somebody@somewhere.com"
   EMailMessage = Range("A1").Value
   EMailSubject = "Тестовое сообщение"
   Application.ScreenUpdating = False
   ActiveSheet.Hyperlinks.Add Anchor:=Range("A1"), Address:= _
      "mailto:" & EMailAddress & _
      "?Subject=" & EMailSubject & _
      "&body=" & EMailMessage
   Range("A1").Hyperlinks(1).Follow NewWindow:=True
   Range("A1").Hyperlinks.Delete
   Application.ScreenUpdating = True
   MsgBox "Нажите кнопку <Отправить> MS Outlook Express"
End Sub
```

Вставка рисунка в рабочий лист из Web

Метод `Insert` семейства `Pictures` позволяет вставлять рисунки не только с локального диска, но и из Интернета. В этом случае в качестве значения параметра метода надо указать URL рисунка, как показано в листинге 20.6.

Листинг 20.6. Вставка рисунка в рабочий лист из Web

```
Sub InsertPicture()
   ActiveSheet.Pictures.Insert "http://www.books.ru/img/81270.jpg"
End Sub
```

Создание web-запроса

В данном разделе продемонстрируем то, как создается web-запрос на примере страницы rate.html, в которой содержатся курсы обмена валюты по отношению к доллару. Для создания web-запроса необходимо произвести следующие действия:

1. Выберите команду **Данные | Импорт внешних данных | Создать Web-запрос**. На экране отобразится окно **Создание веб-запроса** (рис. 20.1).

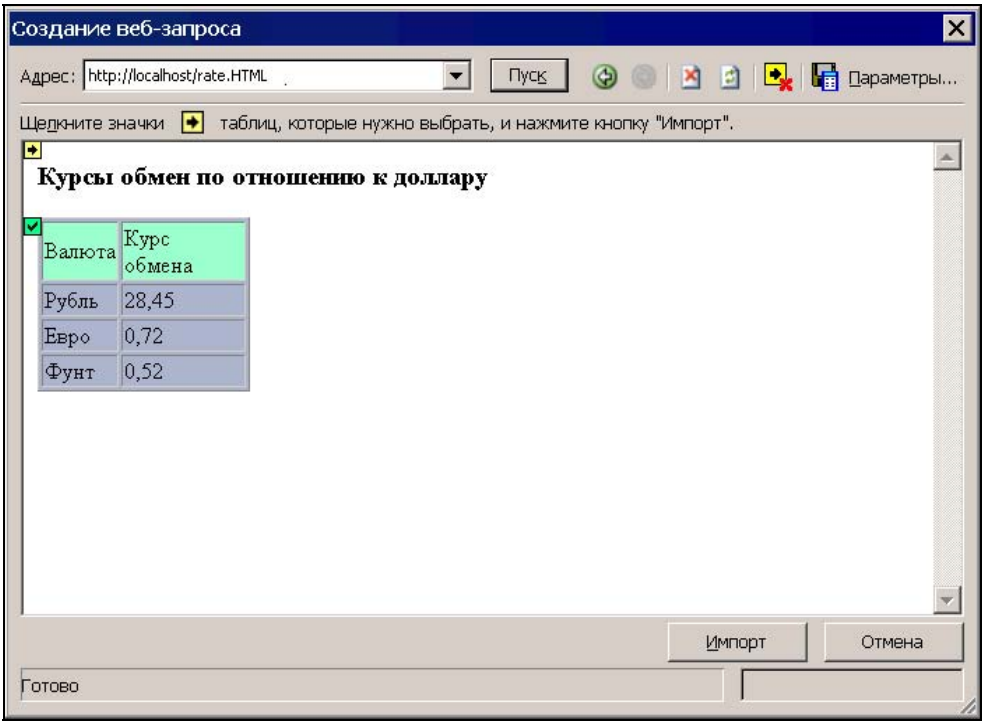


Рис. 20.1. Окно **Создание Web-запроса**

2. Поле **Адрес** предназначено для ввода URL искомой Web-страницы. Например, если на вашем компьютере установлен IIS, то его главным каталогом является **C:\Inetpub\wwwroot**. Расположите в каталоге **C:\Inetpub\wwwroot** файл rate.html. В поле **Адрес** окна браузера нельзя указывать ссылку

C:\Inetpub\wwwroot\rate.html. Вместо этого надо обратиться к файлу через сервер **http://localhost/rate.html** или **http://127.0.0.1/rate.html**. Проще говоря, не забудьте воспользоваться своим сервером. Нажмите кнопку **Пуск**. В результате Web-страница Table.asp загрузится в окно **Создание веб-запроса**.

3. Используя кнопки со стрелками, выделите на Web-странице искомую таблицу.
4. Нажмите кнопку **Импорт**. Отобразится окно **Импорт данных** (рис. 20.2). Группа переключателей **Куда следует поместить данные?** состоит из двух возможностей: **Имеющийся лист** и **Новый лист**. Выберите переключатель **Имеющийся лист** и в соответствующее поле введите ссылку на ячейку (например, **A1**), в которой будет располагаться верхний левый угол импортируемой таблицы.

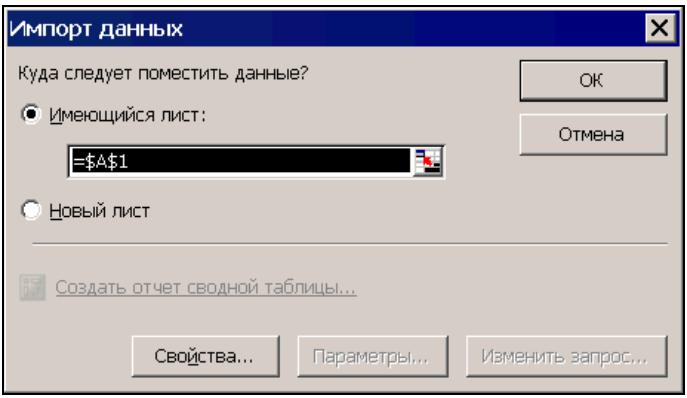


Рис. 20.2. Окно **Импорт данных**

5. Для того чтобы задать параметры запроса, нажмите кнопку **Свойства**. На экране отобразится окно **Свойства внешнего диапазона** (рис. 20.3). Поле **Имя** задает имя запроса. Группа **Определение запроса** задает пароль и устанавливает, надо ли сохранять определение запроса. Группа **Обновление экрана** задает режим обновления импортированных данных. Группа **Формат и разметка данных** устанавливает параметры форматирования. Группа **Если количество строк в диапазоне изменится** задает алгоритм действия при изменении размеров диапазона. Флажок **заполнить формулами соседние столбцы** определяет, надо ли вводить вместе с данными и пояснительные формулы. Оставьте все предлагаемые по умолчанию параметры и нажмите кнопку **ОК**.

Окно **Свойства внешнего диапазона** закроется. Нажмите кнопку **ОК** окна **Импорт данных**. Данные из выбранной таблицы Web-страницы будут импортированы на рабочий лист (рис. 20.4).

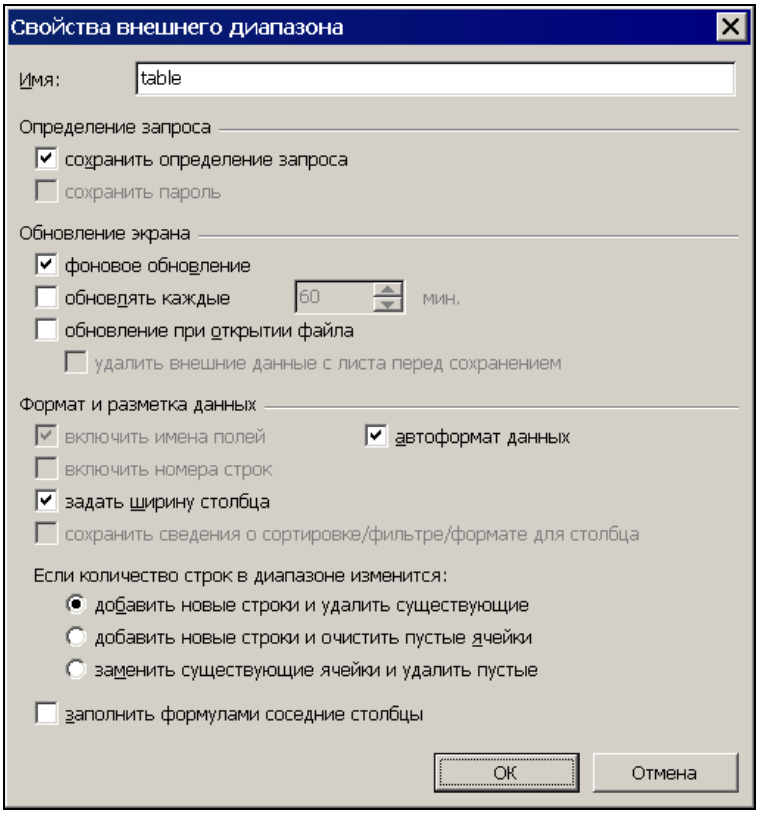


Рис. 20.3. Окно **Свойства внешнего диапазона**

	A	B
1	Валюта	Курс обмена
2	Рубль	28,45
3	Евро	0,72
4	Фунт	0,52
5		

Рис. 20.4. Таблица, импортированная на рабочий лист в результате Web-запроса

Получение данных из Интернета

Данные из Интернета можно получить, загрузив искомую страницу непосредственно в Excel, а затем уже в нем произвести ее обработку. Предположим, что имеется страница rate.html из предыдущего раздела, содержащая

таблицу курса валют по отношению к доллару. Требуется определить курс обмена рублей на доллары. Для этого достаточно методом Open семейства Workbooks загрузить эту страницу в отдельную книгу, а затем методом Find найти ячейку, в которой присутствует слово Рубль. В соседней ячейке справа и будут находиться искомые данные (листинг 20.7).

Листинг 20.7. Получение данных из Интернет с загрузкой всей страницы

```
Sub GetCurrencyRate()
   Dim wb As Workbook
   Dim rgn As Range
   Set wb = Workbooks.Open("http://localhost/rate.html")
   Set rgn = wb.Worksheets(1).Cells.Find("Рубль")
   MsgBox "Курс к доллару : " & rgn.Offset(RowOffset:=0, _
                                    ColumnOffset:=1).Value
End Sub
```

Получение данных из Интернета по запросу из указанной таблицы

Объект QueryTable инкапсулирует в себе данные о web-запросе. Его можно использовать для автоматизации подобного запроса и, кроме того, он позволяет получить данные не со всей страницы, а только из выбранных таблиц. Например, нам надо вывести в рабочую книгу содержание таблицы страницы rate.html, содержащей курсы обмена валют (листинг 20.8). Для этого достаточно методом Add в семейство QueryTables добавить новый элемент, причем надо указать ссылку на ячейку, расположенную в верхнем левом углу того диапазона, куда будет выводиться результат запроса. Далее при помощи свойств QueryTable надо задать параметры запроса. Упомянем только одно из них — WebTables — которое задает номер или номера отображаемых таблиц. Метод Refresh связывается с источником данных.

Листинг 20.8. Получение данных из Интернета по запросу из указанной таблицы

```
Sub GetRatesByWebQuery()
   Dim wb As Workbook
   Dim qt As QueryTable
   Set wb = Workbooks.Add
   With wb.Worksheets(1)
      Set qt = .QueryTables.Add( _
```

```
        Connection:="URL;http://localhost/rate.HTML", _
        Destination:=.Range("A1"))
    End With
    With qt
        .Name = "Курс валют"
        .WebDisableDateRecognition = True
        .RefreshOnFileOpen = False
        .WebFormatting = xlWebFormattingNone
        .BackgroundQuery = True
        .WebSelectionType = xlSpecifiedTables
        .WebTables = "1"
        .SaveData = True
        .AdjustColumnWidth = True
        .Refresh BackgroundQuery:=False
    End With
End Sub
```

Web-компоненты

В MS Office для Web имеются четыре компоненты, представляющие собой ActiveX элементы:

- **Microsoft Office PivotTable 11.0** — для конструирования сводных таблиц;
- **Microsoft Office Chart 11.0** — для построения диаграмм;
- **Microsoft Office Spreadsheet 11.0** — для конструирования электронной таблицы в виде рабочего листа;
- **Microsoft Office DataSource 11.0** — реализует интерфейс для связи Web-страницы с источником данных.

Для использования таких элементов в проектах надо указать на них ссылку. С этой целью:

- выберите команду **Tools | Additional Controls**;
- в окне **Additional Controls** выберите искомый элемент и нажмите кнопку **OK**. Значок элемента будет добавлен в панель инструментов **Toolbox**.

Эти компоненты могут использоваться как в локальных, так и Web-приложениях. В последнем случае браузер должен быть совместим с этими элементами управления, что предполагает использование MS Internet Explorer, начиная с версии 4.01.

Компонента *Microsoft Office Spreadsheet*

Компонента **Microsoft Office Spreadsheet** по своей модели очень похожа на объект `Workbook`. В качестве примера создадим калькулятор, находящий сумму двух чисел с помощью компоненты **Microsoft Office Spreadsheet** сначала для Windows-приложения, а затем для Web-страницы. Итак, на форме расположите компоненту **Microsoft Office Spreadsheet** и кнопку. С помощью окна **Properties** установите у кнопки значения свойств `Name` и `Caption` равными `cmdOK` и `OK`. В модуле формы наберите код из листинга 20.9. Введенные в ячейки **B1** и **B2** числа будут суммироваться и отображаться в ячейке **B3** (рис. 20.5). Нажатие кнопки **OK** вызовет отображение диалогового окна с содержанием ячейки **B3**.

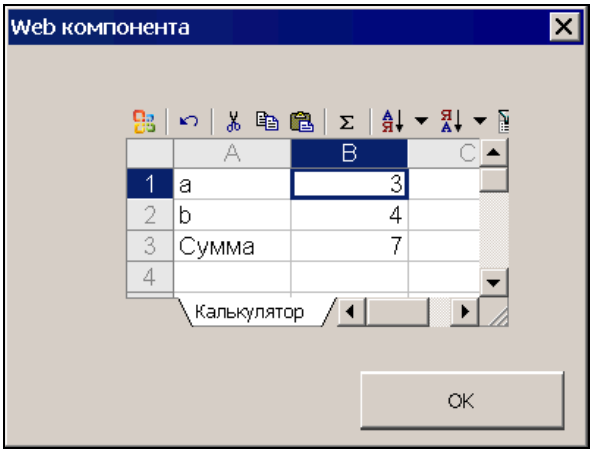


Рис. 20.5. Калькулятор в Windows-приложении, созданный на основе компоненты **Spreadsheet**

Листинг 20.9. Компонента Spreadsheet в Windows-приложении

```
Private Sub UserForm_Initialize()
   Spreadsheet1.Cells(1, 1).Value = "a"
   Spreadsheet1.Cells(2, 1).Value = "b"
   Spreadsheet1.Cells(3, 1).Value = "Сумма"
   Spreadsheet1.Cells(3, 2).Formula = "=B1+B2"
   Spreadsheet1.ActiveSheet.Name = "Калькулятор"
End Sub

Private Sub cmdOK_Click()
   MsgBox "Сумма: " & Spreadsheet1.Cells(3, 2).Value
End Sub
```

Для построения калькулятора на Web-странице, на рабочем листе MS Excel:

- ❒ введите `a` в ячейку **A1**;
- ❒ введите `b` в ячейку **A2**;
- ❒ введите `Сумма` в ячейку **A3**;
- ❒ введите `=B1+B2` в ячейку **B3**;
- ❒ выделите диапазон **A1:A3;B3** и окрасьте его в серый цвет;
- ❒ выберите ярлык рабочего листа, в контекстном меню выберите команду **Переименовать** и установите имя листа равным **Калькулятор**;
- ❒ выделите диапазон **B1:B2** и выберите команду **Формат | Ячейки**. На вкладке **Защита** окна **Формат ячеек** отключите параметр **Защищаемая ячейка**;
- ❒ выберите команду **Сервис | Защита | Защитить лист**. На экране отобразится окно установки пароля. Не вводя пароля, нажмите кнопку **OK**. Таким образом, на рабочем листе создано средство по нахождению суммы двух чисел, причем доступными для пользователя являются только поля для ввода слагаемых;
- ❒ перейдем к созданию Web-страницы. Выделите диапазон **A1:B3**;
- ❒ выберите команду **Сохранить как Web-страницу**;
- ❒ в диалоговом окне **Сохранение документа** выберите переключатель **выделенное $A$1:$B$3**, установите флажок **Добавить итеративность** и нажмите кнопку **Опубликовать**;

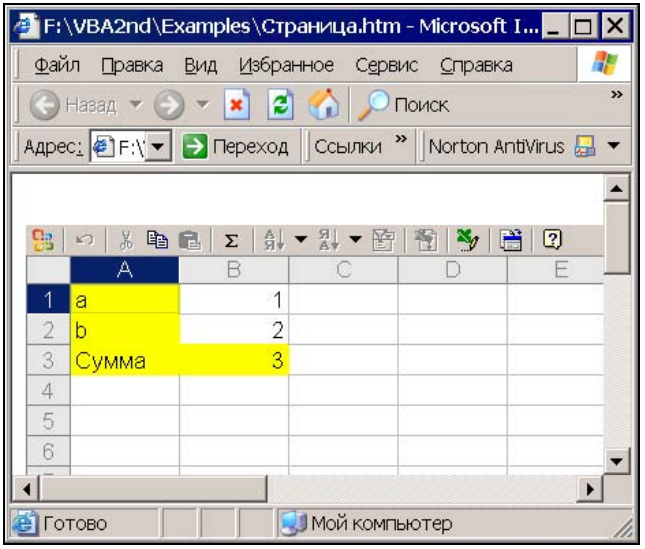


Рис. 20.6. Калькулятор в Web-страницы, созданный на основе компоненты **Spreadsheet**

- в следующем окне в списке **Параметры** выберите **Работа с электронными таблицами**. В поле **Имя** введите имя файла как локальную или как удаленную Web-страницу;
- Нажмите кнопку **Опубликовать**.

В результате в Web-страницу будет вставлен калькулятор, работающий точно так же, как и рабочий лист.

Компонента *Microsoft Office Chart*

Компонента **Microsoft Office Chart** по своей функциональности работает аналогично объекту `Chart`. Продемонстрируем совместную работу компонент **Microsoft Office Spreadsheet** и **Microsoft Office Chart** в Windows-приложении. Создайте форму, на которой расположите эти две компоненты. В модуле формы наберите код из листинга 20.10. Вот и все (рис. 20.7).

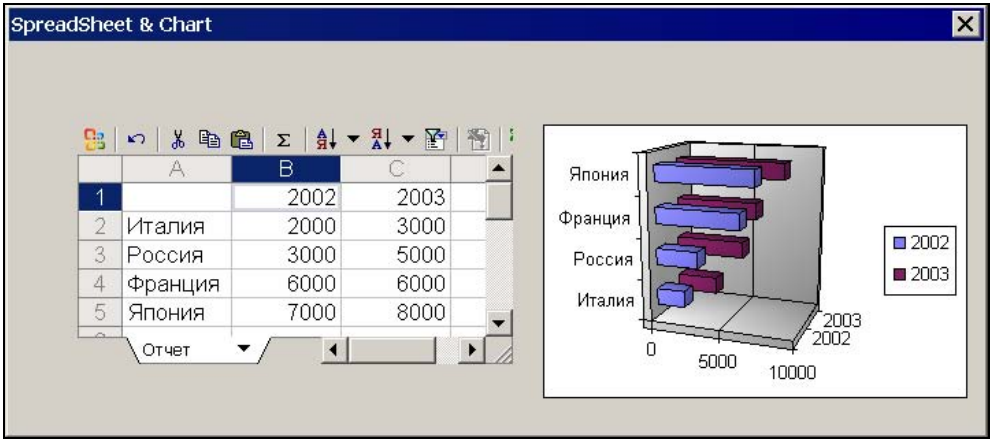


Рис. 20.7. Компонент **Microsoft Office Spreadsheet** и **Microsoft Office Chart** в Windows-приложении

Листинг 20.10. Компонент Spreadsheet и Chart в Windows-приложении

```
Private Sub UserForm_Initialize()
   With Spreadsheet1.Sheets(1)
      .Range("B1:C1").Value = Array(2002, 2003)
      .Range("A2:C2").Value = Array("Италия", 2000, 3000)
      .Range("A3:C3").Value = Array("Россия", 3000, 5000)
      .Range("A4:C4").Value = Array("Франция", 6000, 6000)
```

```
      .Range("A5:C5").Value = Array("Япония", 7000, 8000)
   End With
   Spreadsheet1.Sheets(1).Name = "Отчет"
   With ChartSpace1
      .Charts.Add
      .DataSource = Spreadsheet1
       With .Charts(0)
          .Type = chChartTypeBar3D
          .SeriesCollection.Add
          With .SeriesCollection(0)
             .SetData chDimSeriesNames, 0, "B1"
             .SetData chDimCategories, 0, "A2:A5"
             .SetData chDimValues, 0, "B2:B5"
          End With
         .SeriesCollection.Add
          With .SeriesCollection(1)
             .SetData chDimSeriesNames, 0, "C1"
             .SetData chDimValues, 0, "C2:C5"
          End With
          .HasLegend = True
      End With
   End With
End Sub
```

Web-службы

Web-служба представляет собой программную услугу, предоставляемую пользователю через Интернет. В упрощенном смысле Web-службу можно рассматривать как приложение, сдаваемое пользователю в аренду. От пользователя для его применения и обновления лишь требуется постоянное подключение к Интернету. В MS Office приложениях возможно использование Web-служб. Для этого с сайта Microsoft надо скачать и установить пакет Web Service References Tool 3.0. После чего в выпадающем меню **Tools** редактора Visual Basic появится новая команда **Web Service References**. Для того чтобы применить Web-службу, ее надо сконструировать или найти в сети, затем создать на нее ссылку и после чего применять ее, как если бы она была расположена на вашем компьютере.

Создание Web-службы с помощью Visual Studio .NET

В качестве примера создадим Web-службу с помощью Visual Studio .NET, которая предоставляет в распоряжение пользователя два метода — нахождение квадрата числа и суммы двух чисел. Для этого:

- запустите Visual Studio .NET;
- выберите команду **File | New | Project**;
- в окне **New Project** в списке **Project Types** выберите **Visual Basic Projects**. В списке **Templates** выберете **ASP .NET Web Service**. В поле **Location** введите **http://localhost/MyWebService** (рис. 20.8). Нажмите кнопку **OK**;

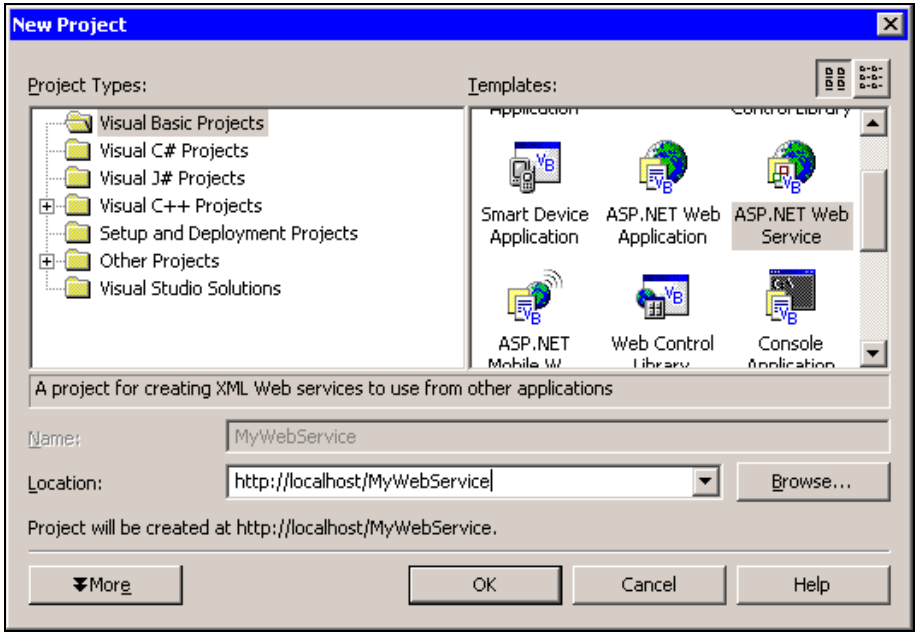


Рис. 20.8. Окно **New Project**

- по умолчанию мастер проекта создаст страницу с именем Service1.asmx. Используя контекстное меню в окне **Solution Explorer,** переименуйте ее в ListFunctions.asmx. Нажмите ссылку **click here to switch to code view**;
- откроется окно кода. В нем будет код, созданный мастером проекта (рис. 20.9 и листинг 20.11). В нем в закомментированном виде приводится пример метода Web-службы `HelloWorld`, который возвращает строку приветствия `Hello World`. Обратите внимание на то, что те методы, которые достижимы пользователю через Web-службу, должны быть помечены атрибутом `<WebMethod()>`;

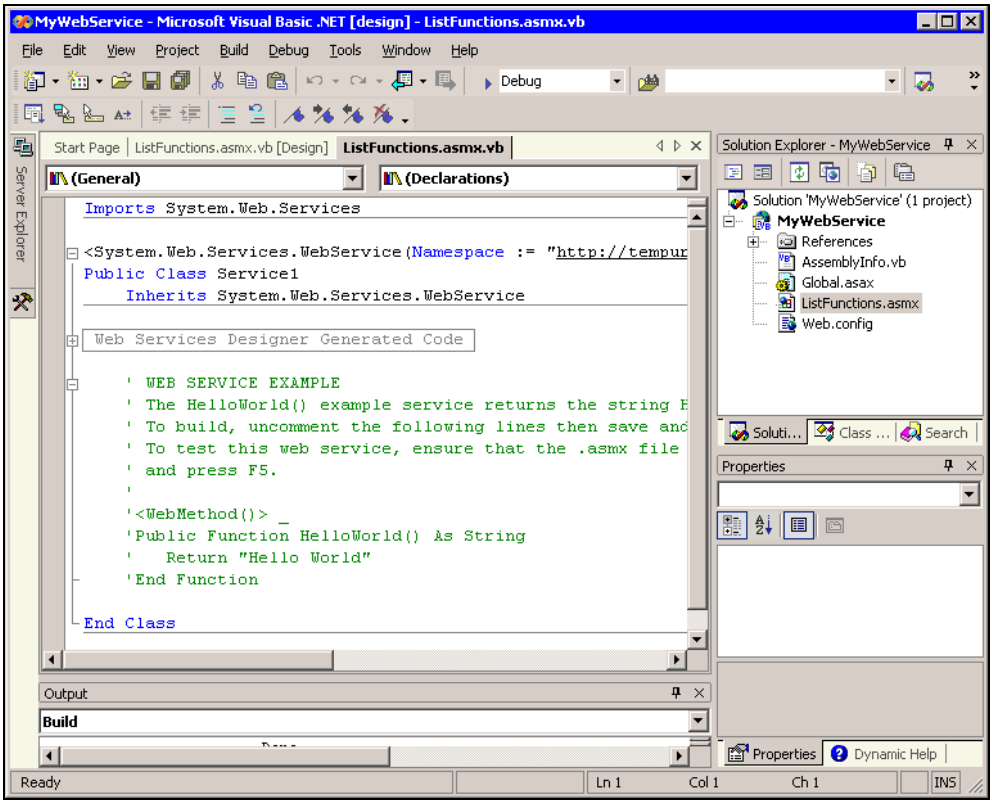


Рис. 20.9. Окно дизайнера Web-служб в Visual Studio .NET

Листинг 20.11. Код, созданный мастером проекта

```
Imports System.Web.Services

<System.Web.Services.WebService( _
Namespace := "http://tempuri.org/MyWebService/Service1")> _
Public Class Service1
    Inherits System.Web.Services.WebService
    ' WEB SERVICE EXAMPLE
    ' The HelloWorld() example service returns the string Hello World.
    ' To build, uncomment the following lines then save
    ' and build the project.
    ' To test this web service,
    ' ensure that the .asmx file is the start page
    ' and press F5.
    '
```

```
    '<WebMethod()> _
    'Public Function HelloWorld() As String
    '   Return "Hello World"
    'End Function
End Class
```

- переименуйте класс, а также измените значение параметра `Namespace` атрибута `System.Web.Services.WebService` с `Service1` на `ListFunctions`. Удалите закомментированную часть кода, а в класс вставьте два Web-метода `MySqr` и `MyAdd` (листинг 20.12);

Листинг 20.12. Измененный код

```
Imports System.Web.Services

<System.Web.Services.WebService( _
Namespace:="http://tempuri.org/MyWebService/ListFunctions")> _
Public Class ListFunctions
    Inherits System.Web.Services.WebService

    <WebMethod()> _
    Public Function MySqr(ByVal x As Double) As Double
        Return x * x
    End Function

    <WebMethod()> _
    Public Function MyAdd(ByVal x As Double, ByVal y As Double) As Double
        Return x + y
    End Function
End Class
```

- выберите команду **File | Save All**;
- откомпилируйте проект, выбрав команду **Build | Build Solution**.

Web-служба готова. Теперь ее надо применить в MS Office проекте.

Применение Web-службы в MS Office

Применим разработанную Web-службу в MS Office приложении, например в MS Excel:

- создайте новую рабочую книгу;
- в редакторе Visual Basic выберите команду **Tools | Web Service References**;

- на экране отобразится окно **Microsoft Office 2003 Web Services Toolkit**. Установите флажок **Web Service URL** и в поле **URL** введите ссылку на Web-службу, в данном случае **http://localhost/mywebservice/ListFunctions.asmx**. Нажмите кнопку **Search**. В списке **Search result** отобразится результат поиска: имя Web-служб с входящими в них методами (рис. 20.10);

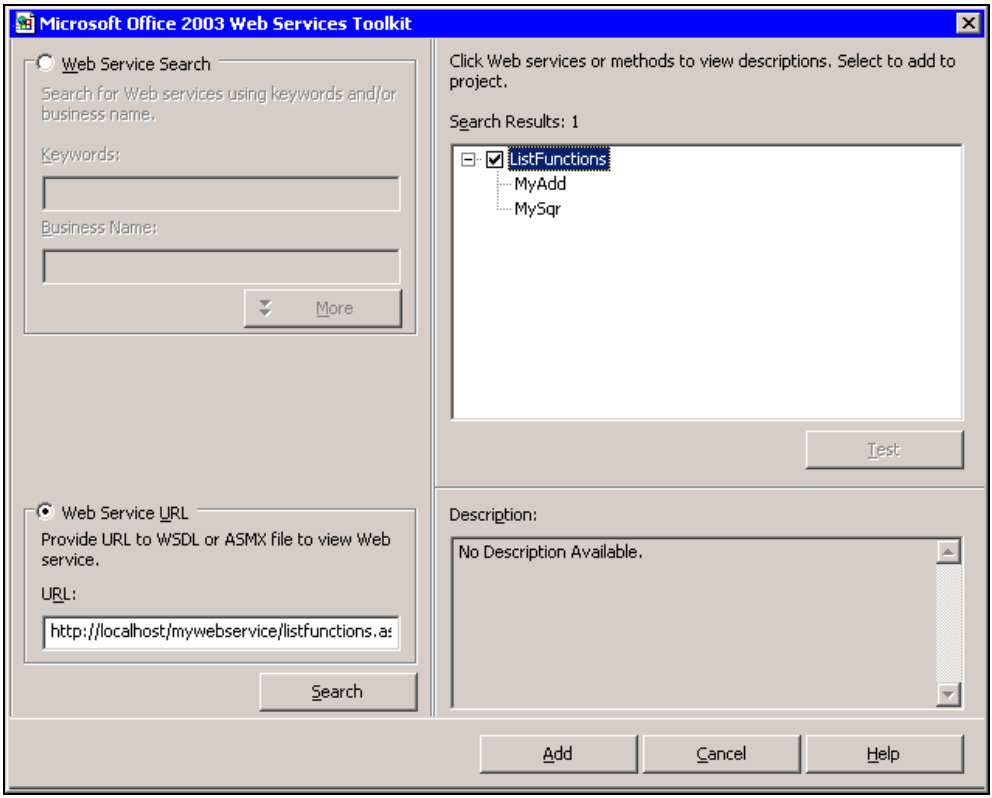


Рис. 20.10. Окно **Microsoft Office 2003 Web Services Toolkit** с результатами поиска Web-службы по указанному адресу.

- отметьте службу `ListFunctions` флажком и нажмите кнопку **Add**. В результате в проект добавится прокси-класс `clsws_ListFunctions`, в котором описаны два метода `wsm_MySqr` и `wsm_MyAdd` (листинг 20.13), реализующие соответствующие методы службы непосредственно на вашем компьютере;

Листинг 20.13. Методы из прокси-класса, созданные мастером проекта

```
Public Function wsm_MySqr(ByVal dbl_x As Double) As Double
    On Error GoTo wsm_MySqrTrap
    wsm_MySqr = sc_ListFunctions.MySqr(dbl_x)
```

```
Exit Function
wsm_MySqrTrap:
    ListFunctionsErrorHandler "wsm_MySqr"
End Function

Public Function wsm_MyAdd(ByVal dbl_x As Double, _
                          ByVal dbl_y As Double) As Double
   On Error GoTo wsm_MyAddTrap
   wsm_MyAdd = sc_ListFunctions.MyAdd(dbl_x, dbl_y)
Exit Function
wsm_MyAddTrap:
    ListFunctionsErrorHandler "wsm_MyAdd"
End Function
```

- добавьте в проект модуль, в который введите код из листинга 20.14, тестирующий Web-службу.

Листинг 20.14. Тестирование Web-службы

```
Public Sub DemoWebService()
   Dim lstFuns As New clsws_ListFunctions
   Dim x As Double, y As Double, z As Double
   x = 2
   z = lstFuns.wsm_MySqr(x)
   Debug.Print CStr(x) & " " & CStr(z)
   x = 1: y = 4
   z = lstFuns.wsm_MyAdd(x, y)
   Debug.Print CStr(x) & " " & CStr(y) & " " & CStr(z)
End Sub
```

Для использования Web-методов на рабочем листе можно определить вспомогательные пользовательские функции, как показано в следующем листинге.

Листинг 20.15. Определение пользовательских функций на основе Web-методов

```
Function Add(x As Double, y As Double) As Double
   Dim lstFuns As New clsws_ListFunctions
   Add = lstFuns.wsm_MyAdd(x, y)
```

```
   Set lstFuns = Nothing
End Function

Function Sqr(x As Double) As Double
   Dim lstFuns As New clsws_ListFunctions
   Sqr = lstFuns.wsm_MySqr(x)
   Set lstFuns = Nothing
End Function
```

Конвертация валют

В качестве еще одного примера воспользуемся достижимой для пользователей всемирной сети Web-службой конвертации указанной суммы денег из одной валюты в другую:

http://webcontinuum.net/webservices/ccydemo.wsdl

У этой службы существует метод, который и производит пересчет денег. Тип валюты задается при помощи трехбуквенного кода aud, cad, eur, gbp, usd или jpy. Итак, подсоедините эту службу к рабочей книге. В результате в проект добавится класс `clsws_ccydemo`, у которого метод `wsm_calcExcRate` осуществляет требуемый пересчет. Этот метод имеет следующий синтаксис:

```
Function wsm_calcExcRate(ByVal str_sCurrIn As String, _
                         ByVal str_sCurrOut As String, _
                         ByVal sng_fAmt As Single) As Single
```

Здесь `str_sCurrIn` — код исходной валюты, `str_sCurrOut` — код получаемой валюты, а `sng_fAmt` — исходная сумма.

В стандартном модуле определите следующую функцию:

```
Function Calc(ByVal str_sCurrIn As String, _
                         ByVal str_sCurrOut As String, _
                         ByVal sng_fAmt As Single) As Single
   Dim obj As clsws_ccydemo
   Set obj = New clsws_ccydemo
   Calc = obj.wsm_calcExcRate(str_sCurrIn, str_sCurrOut, sng_fAmt)
End Function
```

Теперь можно считать. В ячейку **B1** введите код исходной валюты, в ячейку **B2** — получаемой, в **B3** — искомую сумму, а в ячейку **B4** — формулу `=Calc(B1;B2;B3)` (рис. 20.11). Как видите, все, что нужно для расчетов — это подсоединение к Интернету и доступ к Web-службе.

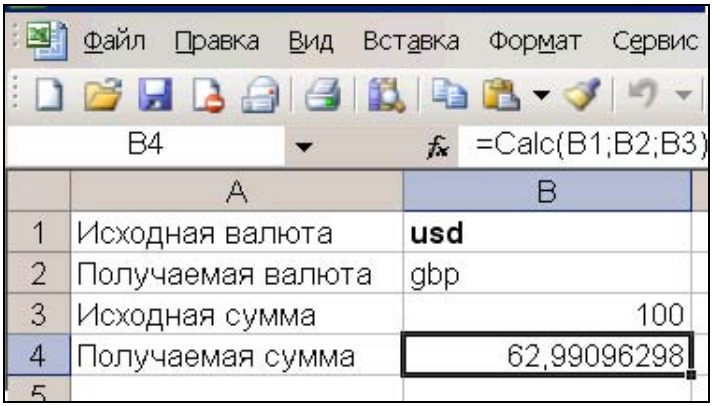
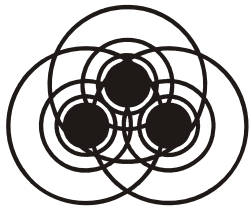


Рис. 20.11. Пересчет валют с помощью Web-службы



Глава 21

VBA и Windows-сценарии

Первый Windows-сценарий

Язык программирования VBScript, являющийся урезанной версией VBA, позволяет писать сценарии как для Web, так и Windows. Windows-сценарии помогают пользователю решать множество рутинных задач типа создания ярлыка на рабочем столе, очистки папок и т. д. Файлы Windows-сценариев являются текстовыми VBS-файлами. Например, создадим Windows-сценарий, который отображает окно с подсказкой дня. В любом текстовом редакторе (например, Notepad) наберите код из листинга 21.1. Сохраните созданный код в файл с расширением vbs, например, Tips.vbs в корневом каталоге диска C:. Поздравляю, ваш первый Windows-сценарий готов. Осталось только протестировать его работоспособность. Для этого в любой программе просмотра (например, в Проводнике) просто щелкните на значке файла. Можно также на рабочем столе создать ярлык, которому назначить следующий объект, где wscript.exe как раз и является той программой, которая проигрывает сценарии:

C:\WINDOWS\wscript.exe c:\ tips.vbs

Листинг 21.1. Совет дня. Файл tips.vbs

```
Dim d(2), n
d(0)="Страшнее кошки зверя нет"
d(1)="Последняя ошибка всегда бывает предпоследней"
d(2)="Если ничего другого не получается, " & vbCr & _
     "то внимательно прочитайте инструкции"
Randomize
n = CInt(2*Rnd())
Msgbox d(n), vbInformation, "Совет дня"
```

Если вызов команды wscript.exe из командной строки кажется слишком утомительным, то в код можно включить инструкции с вызовом соответствующих методов объекта `WScript`, инкапсулирующего в себе средства работы со сценариями (листинг 21.2). В частности, метод `Echo` этого объекта отображает на экране диалоговое окно.

Листинг 21.2. Приветствие. Файл Hello.vbs

```
WScript.Echo "Hello, World!"
```

Метод `Echo` может иметь несколько аргументов. Например,

```
WScript.Echo "Hello, Alice", " and Simon", " and Jim"
```

Доступ к файловой системе

Через функцию `CreateObject` для разработчика достижим объект `FileSystemObject`, что позволяет производить доступ к файловой системе. Например, код из листинга 21.3 отображает в диалоговом окне информацию о дисках: их названиях, метках, свободном и общем объемах.

Листинг 21.3. Сбор данных о дисках

```
Const DrvFixed = 2

Dim objDrive, fs, gb, msg

On Error Resume Next

Set fs = CreateObject("Scripting.FileSystemObject")
gb = 1024^3

msg= ""
For Each objDrive In fs.Drives
   If objDrive.DriveType = DrvFixed Then
      msg= msg & "Drive: " & objDrive.DriveLetter & VbCr
      msg = msg & "Label: " & objDrive.VolumeName & VbCr
      msg = msg & "Free Space: "
      msg = msg & FormatNumber(objDrive.FreeSpace/gb, 2)
      msg = msg & " Gb" & VbCrLf
      msg = msg & "Total Size: "
```

```
      msg = msg & FormatNumber(objDrive.TotalSize/gb, 2)
      msg = msg & " Gb" & VbCr & VbCr
   End If
Next
WScript.Echo msg
Set fs = Nothing
```

Создание документа Word и его печать

Метод `CreateObject` позволяет создавать указанный объект, в частности, приложение Word. После его создания разработчик может работать с ним как с объектом, созданным в обычном VBA приложении. Например, следующий код (листинг 21.4) конструирует новый документ Word, в котором выводит две различно отформатированные строчки текста, после чего этот документ отправляется на печать.

Листинг 21.4. Создание документа Word и его печать

```
Const wdAlignParagraphCenter = 1
Const wdAlignParagraphLeft = 0
Const wdGreen = 11
Const wdRed = 6

Dim objWord, doc, selection
Set objWord = WScript.CreateObject("Word.Application")
Set doc = objWord.Documents.Add
objWord.Visible = True
Set selection = objWord.Selection
selection.Font.Size = 14
selection.ParagraphFormat.Alignment = wdAlignParagraphCenter
selection.Font.Bold = True
selection.Font.ColorIndex = wdGreen
selection.TypeText "Привет всем, кто читает эту книгу!" & vbCrLf
selection.Font.Bold = False
selection.Font.ColorIndex = wdRed
selection.Font.Italic = True
selection.ParagraphFormat.Alignment = wdAlignParagraphLeft
selection.TypeText "А также тем, кто ее еще не читает."
doc.PrintOut
Set doc = Nothing
Set objWord = Nothing
```

Доступ к открытому активному документу Word

Через функцию `CreateObject` для разработчика достижим объект `Word.Application`, а, следовательно, и сам MS Word. Например, код из листинга 21.5 заменяет в открытом документе выделенный фрагмент на текст Hello, Word! полужирного шрифта.

Листинг 21.5. Доступ к открытому активному Word документу

```
Dim objWord

Set objWord = GetObject(, "Word.Application")

objWord.Selection.Font.Bold = True
objWord.Selection.TypeText "Hello, Word!"

Set objWord = Nothing
```

Как определить, запущено ли приложение

При работе с приложениями часто надо определить, запущен ли уже его экземпляр. Следующий код из листинга 21.6 на примере Excel демонстрирует, как это можно сделать.

Листинг 21.6. Как определить, запущено ли приложение

```
Dim obj
Dim msg
On Error Resume Next
Set obj = GetObject(, "Excel.Application")
If Not TypeName(obj) = "Empty" Then
   msg = "Excel Running."
Else
   msg = "Excel Not Running."
End If
MsgBox msg, vbInformation
```

Перевод HTML документа в RTF формат

Для перевода Web-страницы в RTF формат достаточно ее открыть в MS Word, а затем сохранить как RTF. Следующий код (листинг 21.7) демонстрирует такой подход. Он экспортирует документ c:\test.htm в c:\test.rtf.

Листинг 21.7. Перевод HTML документа в RTF формат

```
Const wdFormatRTF=6
Dim objWord
Set objWord = CreateObject("Word.Application")
objWord.Documents.Open ("c:\test.htm")
objWord.Visible = False
objWord.ActiveDocument.SaveAs "c:\test.rtf", wdFormatRTF
objWord.Quit 0
Set objWord = Nothing
```

Доступ к программам и передача им данных с клавиатуры

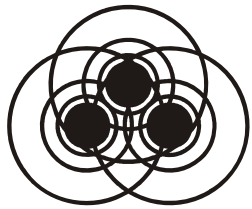
Метод `Exec` объекта `WshShell` позволяет запускать указанные программы на исполнение. Если запущено несколько программ, то их надо уметь идентифицировать. Это можно сделать свойством `ProcessID`. Метод `AppActivate` активизирует приложение с указанным идентификатором. Метод `SendKeys` передает активному приложению комбинацию клавиш, как если бы они были нажаты на клавиатуре. В следующем примере (листинг 21.8) сначала запускается калькулятор. При активном окне калькулятора симулируется нажатие клавиш <2>, <*>, <2>, <Enter>. Затем комбинации клавиш <Ctrl>+<C> для копирования результата вычисления в буфер обмена. Затем комбинацией клавиш <Alt>+<F4> закрываем окно калькулятора и запускаем блокнот. Здесь снова симулируется нажатие клавиш <2>, <*>, <2>, <=>, после чего комбинацией клавиш <Ctrl>+<V> вставляем содержимое буфера обмена. Далее комбинацией клавиш <Alt>+<F4> закрываем окно, причем при помощи последовательного нажатия клавиш <Alt>, <Enter> избегаем сохранения на диск созданного документа.

Листинг 21.8. Передача данных в программы с клавиатуры

```
Dim objShell, objCalculator, objNotepad
Set objShell = WScript.CreateObject("WScript.Shell")
WScript.Echo ("В калькуляторе вычисляем 2*2")
```

```
Set objCalculator = objShell.Exec("calc.exe")
WScript.Sleep 500
objShell.AppActivate objCalculator.ProcessID
objShell.SendKeys "2{*}2~"
WScript.Sleep 500
objShell.SendKeys "^C"
WScript.Echo "Закрываем калькулятор"
objShell.AppActivate objCalculator.ProcessID
objShell.SendKeys "%{F4}"
WScript.Echo "Вставка результата в Блокнот"
Set objNotepad= objShell.Exec("Notepad.exe")
WScript.Sleep 500
objShell.AppActivate objNotepad.ProcessID
objShell.SendKeys "2{*}2="
objShell.SendKeys "^V"
WScript.Sleep 500
objShell.SendKeys "%{F4}"
objShell.SendKeys "{TAB}~"
```

Глава 22



Microsoft Visual Studio Tools для MS Office

Microsoft Visual Studio Tools для Microsoft Office System является новой технологией, позволяющей совместить продуктивность Visual Studio .NET и Microsoft .NET Framework с мощными средствами построения бизнес решений с помощью MS Word и MS Excel. С технической точки зрения это означает, что разработчик, используя Visual Studio .NET 2003 и применяя язык программирования Visual Basic .NET, может писать код, встроенный непосредственно в офисные приложения. Для применения этой технологии на компьютере должны быть установлены Visual Studio .NET 2003, MS Office и Microsoft Visual Studio Tools для Microsoft Office System.

Ваш первый проект в Microsoft Visual Studio Tools

После того, как все требуемые программы установлены:

- запустите Visual Studio .NET;
- выберите команду **File | New | Project**;
- в окне **New Project** (рис. 22.1) в списке **Project Types** выберите **Microsoft Ofiice 2003 Projects | Visual Basic Projects**. В списке **Templates** выберите **Excel Workbook**. В поле **Location** введите местоположение проекта, а в поле **Name** его имя. Нажмите кнопку **OK**;
- на экране отобразится окно **Microsoft Office Project Wizard**, предлагающее создать проект на основе существующей рабочей книги или с нуля (рис. 22.2). Выберите переключатель **Create New Document** для создания проекта с чистого листа и нажмите кнопку **Finish**.

В результате откроется окно разработки приложений с кодом, созданным мастером проекта. В нем нас интересует только одно место, а именно — процедура обработки события `Open` объекта `ThisWorkbook`. В нее добавьте следующий код (листинг 22.1), который обеспечит создание таблицы приветствий в рабочей книге. Нажмите кнопку <F5>, для того чтобы полюбоваться созданной рабочей книгой.

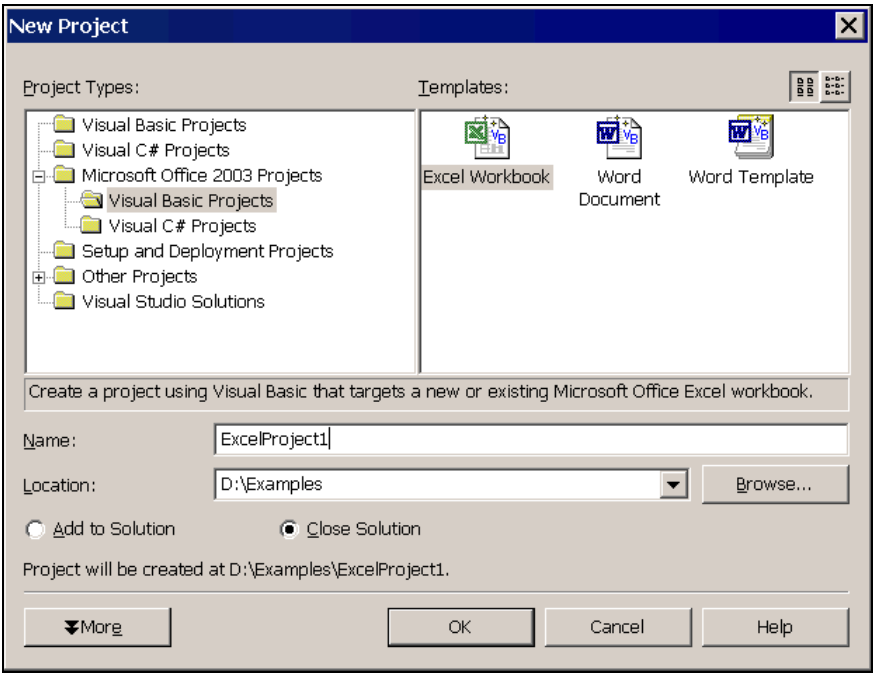


Рис. 22.1. Окно **New Project**

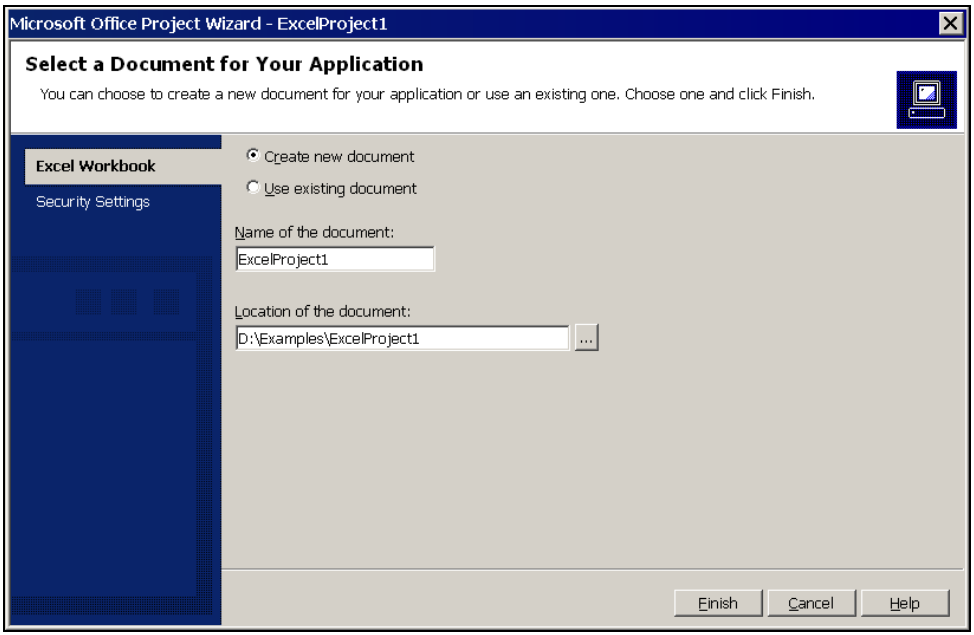


Рис. 22.2. Окно **Microsoft Office Project Wizard**

Листинг 22.1.Рабочая книга с таблицей

```
Private Sub ThisWorkbook_Open() Handles ThisWorkbook.Open
   Dim ws As Excel.Worksheet = New Excel.WorksheetClass
   Dim rg As Excel.Range
   ws = CType(ThisApplication.ActiveWorkbook.ActiveSheet, _
              Excel.Worksheet)
   Dim i As Integer, j As Integer
   For i = 1 To 3
      For j = 1 To 3
         rg = CType(ws.Cells(i, j), Excel.Range)
         rg.Value = "Hello, " & CStr(i) & " " & CStr(j)
      Next
   Next
End Sub
```

В принципе код не очень отличается от того, к чему мы уже привыкли. Единственными особенностями является наличие функции `CType`, которая позволяет явным образом преобразовывать типы данных, объявление процедуры обработки события, в которой при помощи ключевого слова `Handles` производится регистрация обработчика к специфицированному объекту и событию. Дело в том, что в .NET один обработчик может быть зарегистрирован для обработки событий не одного, а нескольких объектов, например, целого массива кнопок.

Перехват событий рабочего листа

У объектов, расположенных в рабочей книге, можно перехватывать из кода события. Продемонстрируем это на примере гиперссылки. Итак, в Visual Studio .NET 2003 создайте новый проект. Запустите его на исполнение и в открывшейся рабочей книге расположите две гиперссылки в ячейках **A1** и **A3**. Сохраните книгу. Нашей целью является то, чтобы при выборе первой гиперссылки отображалось окно с приветствием, а второй — с прощальными словами, и приложение закрывалось. Чтобы это обеспечить, надо обработать событие `SheetFollowHyperlink` объекта `ThisWorkbook`. Итак, используя редактор кода, добавьте в проект процедуру обработки этого события, а в нее введите код из листинга 22.2. Вот и все.

Листинг 22.2. Перехват событий, связанных с выбором гиперссылок

```
Private Sub ThisWorkbook_SheetFollowHyperlink(ByVal Sh As Object, _
   ByVal Target As Microsoft.Office.Interop.Excel.Hyperlink) Handles _
   ThisWorkbook.SheetFollowHyperlink

   If Target.Range.Address = "$A$1" Then
      MsgBox("Hello")
   ElseIf Target.Range.Address = "$A$3" Then
      MsgBox("Bye")
      ThisApplication.Quit()
   End If
End Sub
```

Загрузка данных из базы данных на рабочий лист MS Excel

В качестве еще одного примера применения Microsoft Visual Studio Tools приведем код (листинг 22.3), который загружает из базы данных **Борей** на рабочий лист все записи таблицы **Клиенты** из полей **Название** и **Страна**. При работе с базами данных Microsoft Visual Studio Tools использует механизм ADO .NET, который является развитием ADO для технологии .NET. Для работы данного примера перед описанием созданного мастером проекта класса надо добавить две инструкции импорта пространств имен, в которых находятся классы по работе с базами данных, а именно:

```
Imports System.Data
Imports System.Data.OleDb
```

После этого в код класса, сгенерированного мастером проекта, остается добавить следующие инструкции.

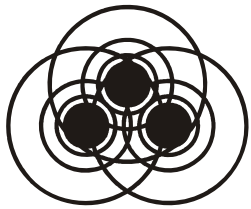
Листинг 22.3. Загрузка данных из базы данных

```
Private Sub ThisWorkbook_Open() Handles ThisWorkbook.Open
   LoadDB()
End Sub

Private Sub LoadDB()
   Try
      Dim strCnn As String = _
```

```
        "Provider=Microsoft.Jet.OLEDB.4.0;Data Source=C:\Борей.mdb"
    Dim cnn As New OleDbConnection(strCnn)
    cnn.Open()
    Dim strSQL As String = _
        "SELECT Название, Страна, Город FROM Клиенты"
    Dim cmd As New OleDbCommand(strSQL, cnn)
    Dim reader As OleDbDataReader = cmd.ExecuteReader
    Dim ws As Excel.Worksheet = ThisWorkbook.ActiveSheet
    ws.Cells(1, 1) = "Название"
    ws.Cells(1, 2) = "Страна"
    Dim n As Integer = 2
    While reader.Read
      ws.Cells(n, 1) = reader(0)
      ws.Cells(n, 2) = reader(1)
      n += 1
    End While
  Catch ex As Exception
    MsgBox(ex.Message)
  End Try
End Sub
```

Бросим быстрый взгляд в данный код. В .NET для работы с базами данных имеются четыре основных класса: `Connection`, `Command`, `DataReader` и `DataAdapter`. Для соединения с базой данных в данном примере используется Microsoft.Jet.OLEDB.4.0 провайдер. Поэтому мы используем вариации этих классов с префиксом `OleDb`, а именно `OleDbConnection`, `OleDbCommand`, `OleDbDataReader` и `OleDbDataAdapter`. Эти классы содержатся в пространстве имен `System.Data.OleDb`. Класс `OleDbConnection` позволяет установить соединение с базой данных, класс `OleDbCommand` — задать SQL команду, а класс `OleDbDataReader` — задать выходной поток, в который выведется результат запроса этой команды.

Глава 23



VBA и XML

Расширенный язык разметки (eXtensible Markup Language, XML) был создан в 1996 г. консорциумом W3C (World Wide Web Consortium) как *метаязык,* предназначенный для описания языков разметки. В языках разметки, подобных HTML, имеются *теги* для структурирования данных. О языке HTML вы можете прочитать в книге "Excel, VBA и Internet в экономике и финансах", опубликованной издательством "BHV-Петербург". Многие называют XML заменой HTML, но это совсем не так. В то время как HTML содержит фиксированный набор тегов, в XML теги вообще отсутствуют. Вместо этого XML позволяет самому разработчику создавать такой язык разметки, который в точности соответствует требованиям конкретного приложения. В приложениях XML обычно используются следующие файлы:

- сам XML-документ, имеющий строго определенную структуру, представляет статический снимок набора данных;
- файл схем (XSD-файл), созданный при помощи XML Schema Definition language языка, который позволяет специфицировать синтаксис XML-документ документа и типа хранимых в нем данных;
- файл стилей (XSL-файл), созданный средствами eXtensible Stylesheet Language языка, который содержит информацию о том, как данные должны быть отформатированы при выводе XML-документа.

Синтаксис XML-документа

Синтаксис XML очень прост. Рассмотрим его на примере кода из листинга 23.1, в котором собраны данные о сотрудниках некоторой фирмы.

Листинг 23.1. Файл List.xml

```
<?xml version="1.0" encoding="windows-1251" ?>
<ListEmployee>
   <Employee>
```

```
      <FullName>
         <FirstName>Джеймс</FirstName>
         <LastName>Бонд</LastName>
      </FullName>
      <Position>
         Представитель
      </Position>
      <Salary>12000</Salary>
      <HireDate>12.10.1989</HireDate>
      <Phone/>
      <Comments/>
   </Employee>
   <Employee>
       <FullName>
          <FirstName>Винни</FirstName>
          <LastName>Пух</LastName>
      </FullName>
      <Position>
         Менеджер
      </Position>
      <Salary>11000</Salary>
      <HireDate>12.11.1991</HireDate>
      <Phone>111-23-12</Phone>
      <Comments> Очень любит мед</Comments>
   </Employee>
</ListEmployee>
```

Первая строка данного листинга — это объявление XML, которое содержит; информацию для анализатора. Эта строка не является обязательной, но, как правило, она присутствует. Атрибут `version="1.0"` указывает версию XML. Атрибут `encoding="windows-1251"` задает используемую в документе кодировку. Следом за объявлением XML идут его элементы. Они состоят из открывающих и закрывающих тегов, например `<LastName>...</LastName>`. Документ XML имеет иерархическую древовидную структуру, в вершине которой находится единственный элемент (узел), называемый корневым. В данном случае узел — это `<ListEmpioyee>`, который включает в себя все остальные элементы. Текст, заключенный между открывающим и закрывающим тегами элемента, называется его содержанием. Например, содержанием элемента `<Comments>Очень любит мед</Comments>` является строка `Очень любит мед`. Элементы могут не иметь содержания. Например, `<Phone></Phone>`. В этом случае их можно объединять в один тег `<Phone/>`.

XML-документы можно отображать непосредственно в окне браузера Internet Explorer, начиная с версии 5. Например, если сохранить созданный XML-документ в файл List.xml, а затем загрузить его в Internet Explorer, то документ будет выглядеть так, как показано на рис. 23.1.

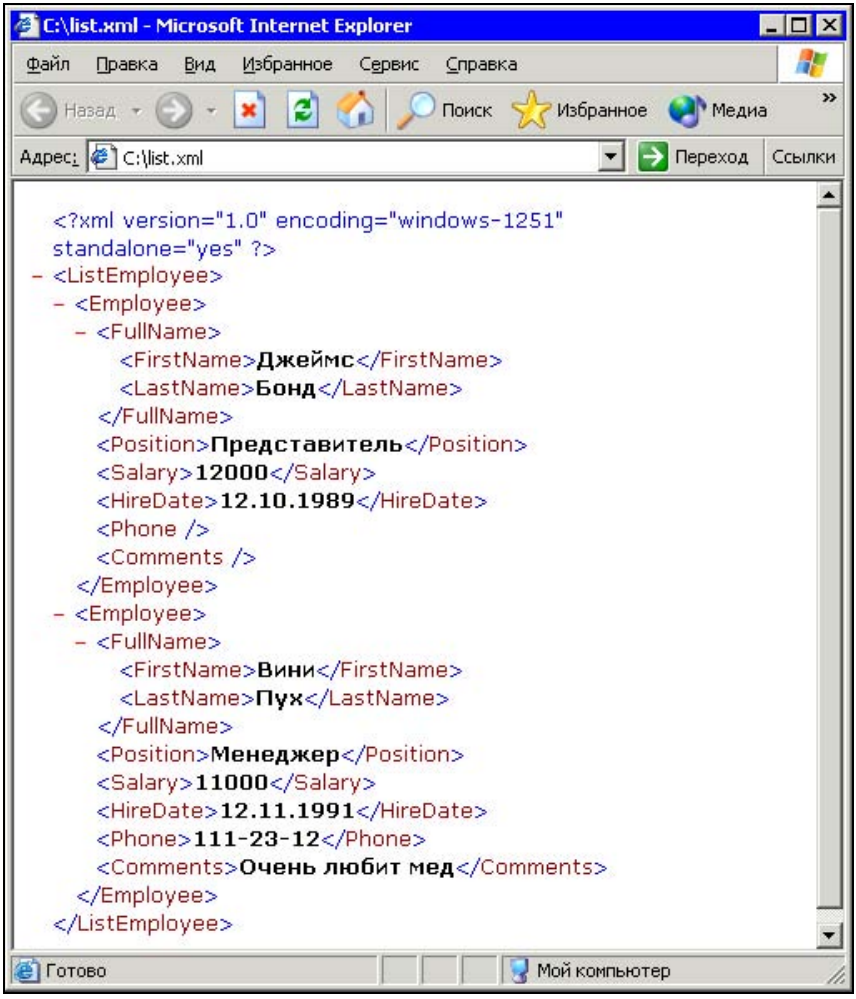


Рис. 23.1. Вид XML-документа в окне Internet Explorer

Синтаксис XSD-документа

После того как мы разобрались с данными, хранящимися в XML-документе, настал момент убедиться, что этот документ имеет определенную структуру, а хранимые в нем данные — специфицированные типы. Здесь, как раз,

на помощь и приходит XSD-документ, представляющий собой XML-схему. В офисных приложениях он позволяет создать карту XML. При добавлении XML-схемы, например, в рабочую книгу, создается объект, называющийся XML-картой. XML-карты используются для создания сопоставленных диапазонов и управления взаимосвязью между этими сопоставленными диапазонами и элементами XML-схемы. Кроме того, эти карты используются для соотнесения содержимого сопоставленного диапазона с элементами схемы при импорте или экспорте XML-данных. Рабочая книга может содержать несколько XML-карт. Каждая XML-карта независима от других, даже если они относятся к одной и той же схеме. XML-карта должна содержать только один корневой элемент. При добавлении схемы, определяющей более одного корневого элемента, пользователя попросят выбрать, какой корневой элемент будет использоваться в новой XML-карте.

Разберем синтаксис XSD-документа на примере кода из листинга 23.2, представляющего собой XML-схему для ранее созданного XML-документа.

Листинг 23.2. Файл list_scheam.xsd

```
<?xml version="1.0" encoding="windows-1251" ?>

<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">

<xsd:element name="ListEmployee" type="ListEmployeeType"/>

<xsd:complexType name="ListEmployeeType">
   <xsd:sequence>
      <xsd:element name="Employee" minOccurs="1" maxOccurs="unbounded"
                   type="EmployeeType"/>
   </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="EmployeeType">
   <xsd:sequence>
      <xsd:element name="FullName" minOccurs="1" maxOccurs="1"
                   type="FullNameType"/>
      <xsd:element name="Position" minOccurs="1" maxOccurs="1"
                   type="PositionType"/>
      <xsd:element name="Salary" minOccurs="1" maxOccurs="1"
                   type="xsd:integer"/>
      <xsd:element name="HireDate" minOccurs="1" maxOccurs="1"
```

```
                    type="xsd:date"/>
      <xsd:element name="Phone" minOccurs="0" maxOccurs="1"
                         type="PhoneType"/>
      <xsd:element name="Comments" minOccurs="0" maxOccurs="1"
                    type="xsd:string"/>
   </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="FullNameType">
   <xsd:sequence>
      <xsd:element name="FirstName" type="xsd:string"/>
      <xsd:element name="LastName" type="xsd:string"/>
   </xsd:sequence>
</xsd:complexType>

<xsd:simpleType name="PositionType">
   <xsd:restriction base="xsd:string">
      <xsd:enumeration value="Менеджер"/>
      <xsd:enumeration value="Представитель"/>
      <xsd:enumeration value="Секретарь"/>
      <xsd:enumeration value="Президент"/>
   </xsd:restriction>
</xsd:simpleType>

<xsd:simpleType name="PhoneType">
   <xsd:restriction base="xsd:string">
      <xsd:pattern value="[0-9][0-9][0-9]-[0-9][0-9]-[0-9][0-9]" />
   </xsd:restriction>
</xsd:simpleType>

</xsd:schema>
```

Первая инструкция данного документа является необязательной. Она задает используемую в документе кодировку. Следующая инструкция определяет пространство имен (namespace), **http://www.w3.org/2001/XMLSchema** и используемый в документе префикс (xsd). Применение пространств имен позволяет избежать конфликта имен.

Следующая инструкция задает элемент `ListEmployee`, как имеющий тип `ListEmployeeType`.

```
<xsd:element name="ListEmployee" type="ListEmployeeType"/>
```

Тип ListEmployeeType определен следующими инструкциями. В них, во-первых, определяется, что тип элемента является составным (complexType), т. е. он может состоять из нескольких элементов Employee, которые имеют тип EmployeeType. Элемент Employee должен присутствовать в документе по крайней мере один раз, что устанавливается атрибутами minOccurs и maxOccurs.

```
<xsd:complexType name="ListEmployeeType">
   <xsd:sequence>
      <xsd:element name="Employee" minOccurs="1" maxOccurs="unbounded"
                   type="EmployeeType"/>
   </xsd:sequence>
</xsd:complexType>
```

Тип EmployeeType имеет более сложную структуру. Он состоит из нескольких элементов: элемента FullName типа FullNameType, который должен присутствовать только один раз; элемента Position типа PositionType, который должен присутствовать только один раз; элемента Salary типа integer, который должен присутствовать только один раз; элемента HireDate типа date, который должен присутствовать только один раз; необязательного элемента Phone типа PhoneType и необязательного элемента Comments типа string.

```
<xsd:complexType name="EmployeeType">
   <xsd:sequence>
      <xsd:element name="FullName" minOccurs="1" maxOccurs="1"
                   type="FullNameType"/>
      <xsd:element name="Position" minOccurs="1" maxOccurs="1"
                   type="PositionType"/>
      <xsd:element name="Salary" minOccurs="1" maxOccurs="1"
                   type="xsd:integer"/>
      <xsd:element name="HireDate" minOccurs="1" maxOccurs="1"
                   type="xsd:date"/>
      <xsd:element name="Phone" minOccurs="0" maxOccurs="1"
                         type="PhoneType"/>
      <xsd:element name="Comments" minOccurs="0" maxOccurs="1"
                   type="xsd:string"/>
   </xsd:sequence>
</xsd:complexType>
```

Тип FullNameType является составным и содержит в себе два элемента типа string. Тип PositionType является простым перечисляемым типом, а тип PhoneType является простым, задающим ввод данных по специфицированному шаблону (в данном случае, "###-##-#", где вместо символа "#" можно использовать любую цифру).

Открытие XML-документа в Excel без использования схемы

Для того чтобы открыть XML-документ в Excel без использования схемы:

- выберите команду **Файл | Открыть**. В окне **Открытие документа** в списке **Тип файла** выберите **Файл XML**. На экране отобразится окно **Открытие XML** (рис. 23.2);

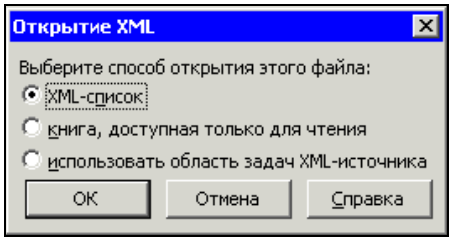


Рис. 23.2. Окно **Открытие XML**

- окно **Открытие XML** позволяет задать способ импорта данных в новую книгу. Переключатель **XML-список** загружает документ в список. Переключатель **книга доступна только для чтения** загружает документ в режиме только для чтения, а переключатель **использовать область задач XML-источника** отображает соответствующую область задач, в которую выводит схему, созданную самим Excel на основе анализа XML-документа. После выбора требуемого режима импорта нажмите кнопку **OK**.

Открытие XML-документа в Excel с использованием схемы

Для открытия XML-документа в Excel с использованием схемы:

- выберите команду **Данные | XML | Источник XML**. На экране отобразится область задач **Источник XML**;
- нажмите кнопку **Карты XML**. На экране отобразится окно **Карты XML**. В этом окне нажмите кнопку **Добавить** и, используя появившееся окно **Выберите источник XML**, укажите ссылку на искомый XSD-файл и нажмите кнопку **Открыть**;
- ссылка на выбранный источник добавится в окне **Карты XML**. Нажмите кнопку **OK**. Схема в виде иерархической структуры отобразится в области задач **Источник XML**;
- можно переместить в ячейки рабочего листа как все вершины, так и только отдельные из них. Например, переместите вершину **LastName**

в ячейку **B1**, вершину **Position** — в ячейку **C1**, вершину **HireDate** — в ячейку **D1** и вершину **Comments** — в ячейку **E1**;

- остается только в соответствии с данной схемой импортировать данные. Для этого выберите команду **Данные | XML | Импорт**. На экране отобразится окно **Источник XML**. В этом окне выберите ссылку на соответствующий файл, нажмите кнопку **Импорт** и данные будут загружены в созданный вами список (рис. 23.3).

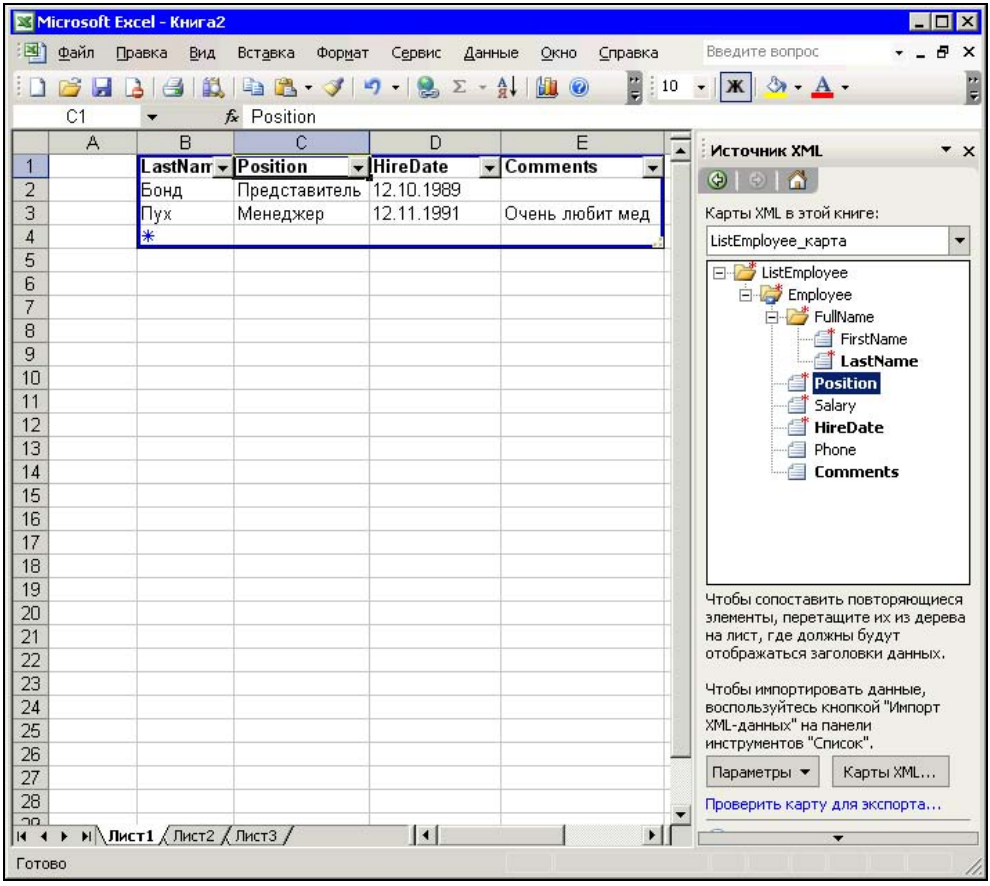


Рис. 23.3. Схема в области **Источник XML** и связанный с ней список с импортированными данными из XML-документа

Импорт XML-карты в коде

Данные о XML-карте инкапсулируются в объекте `XmlMap`. Все карты документа образуют семейство `XmlMaps`, и добавление нового элемента в него в коде производится методом `Add`, как показано в листинге 23.3.

Листинг 23.3. Импорт схемы в коде

```
Sub ImportXMLSchema()
   Dim objMap As XmlMap
   Dim FileName As String
   FileName = "C:\list_scheam.xsd"
   Set objMap = ActiveWorkbook.XmlMaps.Add(xSchemaFile)
End Sub
```

Создание связанных с картой столбцов и импорт в них данных

После того как карта вставлена в книгу, необходимо ее элементы связать со столбцами списка. Здесь на помощь приходит метод `SetValue` объекта `XPath`, ассоциированного со столбцом списка. Этот метод позволяет связать со столбцом вершину карты. Например, в следующем коде (листинг 23.4) в диапазоне **A1:C1** создается список. Со столбцом этого списка с заголовком в ячейке **A1** связывается вершина `FirstName`, со столбцом с заголовком в ячейке **B1** — вершина `LastName`, а со столбцом с заголовком в ячейке **C1** — вершина `Position`. Остается только импортировать данные в связанный список методом `Import` объекта `XmlMap`.

Листинг 23.4. Создание связанных с картой столбцов

```
Private Sub CreateMappedColumns()
   Dim objMap As XmlMap
   Dim strXPath As String
   Dim objList As ListObject
   Dim xmlFile As String
   xmlFile = "C:\list.xml"
   Range("A1:C1").Select
   Set objMap = ActiveWorkbook.XmlMaps(1)
   Set objList = ActiveSheet.ListObjects.Add

   strXPath = "/ListEmployee/Employee/FullName/FirstName"
   objList.ListColumns(1).XPath.SetValue objMap, strXPath
   Range("A1").Value = "First Name"

   strXPath = "/ListEmployee/Employee/FullName/LastName"
```

```
    objList.ListColumns(2).XPath.SetValue objMap, strXPath
    Range("B1").Value = "Last Name"

    strXPath = "/ListEmployee/Employee/Position"
    objList.ListColumns(3).XPath.SetValue objMap, strXPath
    Range("C1").Value = "Position"

    ThisWorkbook.XmlMaps(1).Import xmlFile
    If Err.Number = 0 Then
       MsgBox "Данные " & xmlFile & " импортированы."
    Else
       MsgBox "Ошибка при импорте " & xmlFile
       Exit Sub
    End If
End Sub
```

Если необходимо импортировать не отдельные столбцы, а весь XML-документ, то можно все сделать за одну операцию при помощи метода `XmlImport`. Например, следующая инструкция импортирует документ C:\list.xml с использованием указанной карты так, чтобы левая верхняя вершина списка с импортированными данными находилась в ячейке **A1**.

```
ActiveWorkbook.XmlImport "C:\list.xml", _
                         ActiveWorkbook.XmlMaps(1), , Range("A1")
```

Приложение 1

VBA и игра *Сапер*

В данном приложении мы воспользуемся VBA для развлечений, а именно, на примере популярной игры Сапер продемонстрируем то, как игры можно создавать средствами VBA. Игра Сапер входит в стандартные программы, поставляемые вместе с Windows. Используя VBA и рабочий лист нетрудно самим создать подобную игру (листинги П.1, *а*, П.1, *б*). В нашем случае в качестве игрового поля будут ячейки диапазона **B2:G7**. Кроме того, в нашем распоряжении будет кнопка, которая инициализирует новую игру, а число спрятанных мин отображается в статусной строке (рис. П.1). При щелчке на поле либо отображается количество находящихся в соседних полях мин, либо, при подрыве, мина, которая там была спрятана, окрашенная в красный цвет и, кроме того, окрашиваются все остальные мины. Идея программы очень простая. Первоначально располагаем в клетках игрового поля случайным образом мины, помечая их символом "*". Для того чтобы пользователь их не видел,

Рис. П.1. Игра *Сапер*

они окрашиваются в цвет фона. Затем при выборе пользователем ячейки игрового поля обрабатывается событие `SelectionChange` объекта `SelectionChange` и определяется значение выбранной ячейки. Если оно равно "*", то игра проиграна. В противном случае надо сосчитать, сколько таких звездочек находится в соседних ячейках и это число вывести в выбранной ячейке.

Листинг П.1, *а*. Игра Сапер. Модуль рабочего листа

```
Public numberBomb As Integer


Private Sub cmdNewGame_Click()
   newGame
End Sub

Private Sub Worksheet_SelectionChange(ByVal Target As Range)
   Dim i, j As Integer
   Dim IsIn As Boolean
   IsIn = (Target.Row >= 2) And (Target.Row <= 7) _
           And (Target.Column >= 2) And (Target.Column <= 7)
   numberBomb = 0
   If Target.Value = "*" And IsIn Then
      Target.Font.Color = RGB(0, 0, 0)
      Target.Interior.Color = RGB(255, 0, 0)
      gameOver
   ElseIf IsIn Then
        Target.Interior.Color = RGB(0, 0, 255)
        Target.Font.Color = RGB(0, 255, 0)
        For i = Target.Column - 1 To Target.Column + 1
           For j = Target.Row - 1 To Target.Row + 1
              If Target.Worksheet.Cells(j, i).Value = "*" Then
                 numberBomb = numberBomb + 1
              End If
           Next
        Next
       Target.Value = numberBomb
       Target.Font.Size = 16
    End If
End Sub
```

Листинг П.1, *б*. Игра Сапер. Стандартный модуль

```
Public Sub HelloForm()
   Application.ActiveSheet.Cells.Locked = True
   frmStart.Show
End Sub

Public Sub FirstDo()
   Cells.Interior.Color = RGB(0, 200, 75)
   Cells.Font.Color = RGB(0, 0, 0)
   Cells.Font.Size = 18
   Cells.HorizontalAlignment = xlCenter
   For i = 2 To 7
      For j = 2 To 7
        Cells(i, j).Interior.Color = RGB(200, 200, 200)
        Cells(i, j).Value = ""
     Next
   Next
End Sub
Public Sub gameOver()
   For i = 2 To 7
      For j = 2 To 7
        If Cells(i, j).Value = "*" Then
           Cells(i, j).Font.Color = RGB(0, 0, 0)
        End If
     Next
   Next
   MsgBox "Проигрыш"
End Sub

Public Sub newGame()
   FirstDo
   Dim i As Byte, j As Byte, k As Byte
   Dim nmine As Integer
   For i = 0 To 9
      j = Int((6 * Rnd) + 1) + 1
```

```
    k = Int((6 * Rnd) + 1) + 1
    Cells(j, k).Font.Color = RGB(200, 200, 200)
    Cells(j, k).Value = "*"
Next
nmine = 0
For i = 2 To 7
    For j = 2 To 7
       If Cells(i, j).Value = "*" Then
          nmine = nmine + 1
       End If
    Next
Next
Application.StatusBar = "Число мин " & nmine
```

End Sub

Предметный указатель

Л

М

Н

О

П

Р

С

Т

Ф

Ц

Ш