Microsoft®

Excel® 2010

Профессиональное программирование на VBA

Джон Уокенбах

На компакт-диске:

- файлы примеров для всех упражнений книги
- электронная англоязычная версия книги

Microsoft® Excel® 2010

Профессиональное программирование на VBA

Excel® 2010
Power Programming with VBA

by John Walkenbach

Wiley Publishing, Inc.

Microsoft® Excel® 2010

Профессиональное программирование на VBA

Джон Уокенбах

ДИАЛЕКТИКА
Москва • Санкт-Петербург • Киев
2012

ББК 32.973.26-018.2.75
У62
УДК 681.3.07

Компьютерное издательство "Диалектика"

Главный редактор *С.Н. Тригуб*

Зав. редакцией *В.Р. Гинзбург*

Перевод с английского и редакция *А.П. Сергеева*
По общим вопросам обращайтесь в издательство "Диалектика" по адресу:
info@dialektika.com, http://www.dialektika.com

Уокенбах, Джон.

У62 Excel 2010: профессиональное программирование на VBA. : Пер. с англ. — М. : ООО "И.Д. Вильямс", 2012. — 944 с. : ил. — Парал. тит. англ.

ISBN 978-5-8459-1721-8 (рус.)

ББК 32.973.26-018.2.75

Научно-популярное издание

Джон Уокенбах

Excel 2010: профессиональное программирование на VBA

Литературный редактор *Л.Н. Красножон, Е.Д. Давидян*
Верстка *О.В. Романенко*
Художественный редактор *Е.П. Дынник*
Корректор *Л.А. Гордиенко*

Подписано в печать 06.04.1012. Формат 70x100/16
Гарнитура Times. Печать офсетная
Усл. печ. л. 76,11. Уч.-изд. л. 52,8
Доп. тираж 1000 экз. Заказ № 3107

Первая Академическая типография "Наука"
199034, Санкт-Петербург, 9-я линия, 12/28

ООО "И. Д. Вильямс", 127055, г. Москва, ул. Лесная, д. 43, стр. 1

ISBN 978-5-8459-1721-8 (рус.)

ISBN 978-0-470-47535-5 (англ.)

Оглавление

Содержание

Об авторе

Джон Уокенбах — автор более 50 книг, посвященных электронным таблицам. Живет в штате Аризона. Посетите его веб-сайт `http://spreadsheetpage.com`.

Введение

Рад приветствовать вас, уважаемые читатели! Если вы разрабатываете электронные таблицы, предназначенные для других пользователей, или просто хотите серьезно изучить Excel, то, купив эту книгу, вы сделали правильный выбор.

О чем эта книга

Предмет рассмотрения этой книги — язык программирования Visual Basic for Applications (VBA), который встроен в Excel, а также в другие приложения, входящие в состав Microsoft Office. Здесь подробно описано создание программ, автоматизирующих выполнение различных задач в Excel, а также рассматривается широкий круг других тем — от написания простейших макросов до создания сложнейших приложений и утилит, рассчитанных на взаимодействие с пользователем.

В этой книге нет описания программного пакета Microsoft Visual Studio Tools for Office (VSTO). Он представляет собой воплощение относительно новой технологии, использующей Visual Basic .NET и Microsoft Visual C#. Технология VSTO также может применяться для управления поведением программы Excel и других приложений Microsoft Office.

Что необходимо знать

Книга не предназначена для начинающих пользователей Excel. Если у вас нет опыта работы с этим приложением, то прочтите сначала мою книгу *Excel 2010. Библия пользователя* (Диалектика, 2011), в которой подробно рассказывается обо всех возможностях Excel (она адресована пользователям всех уровней).

Чтобы извлечь из книги максимум полезной информации, необходимо быть достаточно опытным пользователем Excel. Я не уделяю особого внимания выполнению простых операций. Предполагается, что вы умеете следующее:

- создавать рабочие книги, вставлять листы, сохранять файлы и т.д.;
- перемещаться по рабочей книге;
- работать с лентой Excel;
- вводить формулы;
- использовать функции рабочих листов Excel;
- давать имена ячейкам и диапазонам;
- пользоваться основными средствами Windows, такими, например, как буфер обмена и Проводник.

Если вы не знаете, как все это делать, то, возможно, некоторая информация окажется для вас довольно сложной. Итак, считайте, что вас предупредили. Если же вы опытный пользователь электронных таблиц, но еще не работали с Excel 2010, изучите краткий обзор возможностей этой программы в главе 2.

Что вам понадобится

Для эффективной работы с книгой нужно иметь копию Excel 2010. Несмотря на то что большая часть сведений, изложенных в книге, применима к Excel 2003 и 2007, предполагается, что вы работаете именно с Excel 2010. И хотя программы Excel 2007 и Excel 2010 радикальным образом отличаются от своих предшественниц, среда разработки VBA не слишком изменилась. Если вы планируете разрабатывать приложения, которые ориентированы на более ранние версии Excel, не используйте в качестве среды разработки Excel 2010.

Кроме того, материал книги в основном применим и к Excel для Macintosh. Впрочем, я не проводил испытаний на совместимость с версией для Mac (оставляю это для вас).

Достаточно иметь компьютерную систему на основе платформы Windows, но лучше приобрести компьютер с большим объемом оперативной памяти. Excel — сложная программа, и ее использование в низкопроизводительной системе или в системе с небольшим объемом памяти не доставит вам удовольствия.

Рекомендуется установить разрешение экрана 1280×1024 пикселей, а еще лучше — 1600×1200 пикселей. Для достижения оптимальных результатов воспользуйтесь системой на основе двух мониторов, на одном из которых будет отображаться окно программы Excel, а на другом — окно редактора Visual Basic Editor.

Для просмотра примеров на прилагаемом к книге компакт-диске понадобится привод для чтения компакт-дисков.

Соглашения, используемые в книге

Найдите время, чтобы просмотреть этот раздел и узнать о некоторых соглашениях, используемых по всей книге.

Команды Excel

После версии Excel 2007 на сцену вышел новый, лишенный меню пользовательский интерфейс. На смену прежней системе меню пришла контекстная система на основе ленты. Вдоль верхнего края ленты отображаются названия *вкладок* (например, Вставка (Insert), Вид (View) и т.д.). После щелчка мышью на вкладке отображаются пиктограммы, соответствующие кнопкам команд данной вкладки. Каждой пиктограмме соответствует имя, которое обычно отображается правее или ниже значка пиктограммы. Совокупность пиктограмм образует *группы*, ниже которых отображаются их имена.

В соответствии с соглашением, используемым в данной книге, сначала указывается название вкладки, за которым следует имя группы, а завершает все это название пиктограммы. Например, команда, определяющая перенос слов в ячейке таблицы, записывается следующим образом:

Главная⇨Выравнивание⇨Перенос текста (Home⇨Alignment⇨Wrap Text).

После выбора первой вкладки, называемой Файл (File), на экране появляется окно представления Backstage. В левой его части отображается ряд вкладок, включающих различные команды для работы с файлами, печати и т.д. Для обозначения команд, находящихся в окне представления Backstage, применяется слово "Файл", за которым следует название команды. Например, после вызова на выполнение следующей команды отображается диалоговое окно Параметры Excel (Excel Options):

Файл⇨Параметры Excel (File⇨Excel Options).

Команды редактора VBA

Редактор VBA представляет собой окно, в котором осуществляется работа с кодом VBA. В этом окне применяется традиционный интерфейс, предусматривающий использование меню и панелей инструментов. Например, ниже приведена команда, суть которой заключается в выборе меню Tools (Сервис) с последующим выбором элемента меню References (Ссылки):

Tools⇨References.

Ввод данных с клавиатуры

Все, что вводится с клавиатуры, отображается полужирным шрифтом, например "введите **=СУММ(B2:B50)** в ячейку B51".

Длинное вводимое значение располагается в отдельной строке, и для него используется моноширинный шрифт. Например, вы можете получить указание ввести следующую формулу:

```
=VLOOKUP(StockNumber,PriceList,2)
```

Код VBA

В этой книге вы будете сталкиваться с фрагментами кода VBA, а также с листингами процедур. В каждом листинге используется моноширинный шрифт, а каждая строка кода занимает в тексте книги отдельную строку. Чтобы код было удобнее читать, используются отступы. Конечно, задавать отступы не обязательно, но они помогают отделять друг от друга операторы, находящиеся рядом.

Если строка кода не помещается в одной строке книги, то используется стандартный для VBA метод продолжения строки: в конце строки вводится пробел, после которого располагается символ подчеркивания. Это означает, что данная строка кода продолжается следующим фрагментом. Например, приведенные ниже две строки текста представляют единую строку кода.

```
If Right(ActiveCell, 1) = "!" Then _
  ActiveCell = Left(ActiveCell, Len(ActiveCell) - 1)
```

Этот код можно ввести или так, как показано выше (т.е. в двух строках), или в одной строке, не используя символ подчеркивания.

Функции, имена файлов и именованные диапазоны

Для отображения функций рабочих листов Excel используется верхний регистр моноширинного шрифта (например, "В ячейку C20 введите формулу **СУММ**"). Имена, свойства, методы и объекты процедур VBA отображаются моноширинным шрифтом ("Выполните процедуру `GetTotals`"). В таких именах, чтобы они легче читались, нередко одновременно используются и нижний, и верхний регистры.

Моноширинный шрифт также применяется для обозначения названий файлов и именованных диапазонов ("Откройте файл `myfile.xlsm` и выберите именованный диапазон").

Что означают пиктограммы

По всей книге слева от абзацев можно встретить пиктограммы. Они используются для того, чтобы привлечь ваше внимание к особо важной информации.

Новинка

Эта пиктограмма обозначает новые возможности, которые появились в Excel 2010.

Примечание

Я пользуюсь этой пиктограммой, чтобы подчеркнуть важность принципа, который поможет вам легко справиться с задачей или понять изложенную далее информацию.

Совет

Эта пиктограмма указывает на более эффективный способ выполнения чего-либо или на прием, который, возможно, не является очевидным.

Компакт-диск

Этой пиктограммой обозначаются описанные в книге примеры, которые можно найти на прилагаемом компакт-диске (см. приведенный далее раздел "О компакт-диске").

Предупреждение

Эта пиктограмма используется для описания операции, которую следует выполнять осторожно во избежание возможных проблем.

Перекрестная ссылка

Эта пиктограмма используется для того, чтобы предоставить ссылки на другие главы, в которых о том или ином предмете говорится более подробно.

Структура книги

Главы книги сгруппированы в восемь частей.

Часть I. Введение в Excel

В этой части представлены основы изучаемой программы. В главе 1 излагается краткая история электронных таблиц. Таким образом, вы сможете определить место Excel в мире программного обеспечения. В главе 2 предложен концептуальный анализ Excel 2010 — достаточно полезный для тех опытных пользователей электронных таблиц, которые только переходят к использованию Excel. Что же касается главы 3, то в ней вкратце рассматриваются формулы и, кроме того, рассказывается о некоторых новых приемах. В главе 4 отмечены достоинства и недостатки разных форматов файлов, поддерживаемых и создаваемых в Excel.

Часть II. Разработка приложений Excel

Данная часть включает две главы. В главе 5 в общих чертах рассматривается создание приложений электронных таблиц. В главе 6 этот вопрос освещен более глубоко: в ней описаны типичные действия по разработке приложений электронных таблиц.

Часть III. Visual Basic for Applications

В эту часть входят главы 7–11. Здесь речь идет о подготовке к изучению VBA. Вы ознакомитесь с VBA, с основами программирования и более подробно — с разработкой процедур и функций VBA. В главе 11 предлагаются примеры использования VBA для решения повседневных задач.

Часть IV. Пользовательские формы

В этой части речь идет о пользовательских диалоговых окнах. В главе 12 описываются альтернативные методы создания пользовательских диалоговых окон. О пользовательских формах и различных элементах управления, используемых при создании этих форм, рассказывается в главе 13. В главах 14 и 15 приведены примеры пользовательских диалоговых окон, начиная от простых и заканчивая достаточно сложными.

Часть V. Профессиональные методы программирования

В этой части рассматриваются дополнительные методы программирования, которые, на первый взгляд, невероятно сложны для понимания. Из первых трех глав вы узнаете, как создавать утилиты и использовать VBA для работы со сводными таблицами и диаграммами, а также со спарклайнами, которые появились в Excel 2010. В главе 19 речь идет о процессе *обработки событий*, который позволяет выполнять процедуры автоматически, причем именно тогда, когда происходят определенные события. Из главы 20 вы узнаете о способах взаимодействия с другими приложениями (такими, например, как Word). Часть V заканчивается главой 21, в которой подробно описаны способы создания надстроек.

Часть VI. Разработка приложений

Эта часть посвящена важным этапам создания приложений, ориентированных на конечных пользователей. В главе 22 рассматриваются методы изменения ленты, в главе 23 обсуждается работа с контекстными меню Excel, в главе 24 описываются способы подготовки интерактивной справочной системы для приложений и в главе 25 приводятся сведения о разработке приложений, ориентированных на пользователей.

Часть VII. Дополнительные темы

В этой части освещаются дополнительные темы, которые будут вам полезны. Информация о совместимости приведена в главе 26. В главе 27 обсуждаются способы применения VBA для работы с файлами. Что же касается главы 28, то в ней объясняется, как с помощью VBA управлять такими компонентами Visual Basic, как пользовательские диалоговые окна и модули. В главе 29 речь идет о модулях классов. Глава 30 посвящена методам работы с цветом в Excel. Заканчивается данная часть полезной главой 31, в которой приводятся ответы на многие часто задаваемые вопросы о программировании в Excel.

Часть VIII. Приложения

Завершают книгу четыре приложения. В приложении А вы найдете полезную информацию о ресурсах Интернета, посвященных Excel. Приложение Б — это справочное руководство по всем ключевым словам VBA (операторам и функциям). Коды ошибок VBA

приведены в приложении В. Наконец, в приложении Г описаны файлы, которые содержатся на прилагаемом к книге компакт-диске.

О компакт-диске

К данной книге прилагается компакт-диск с файлами примеров, рассматриваемых в книге. Я убежден, что любые знания лучше получать, изучая конкретные примеры, которые могут пригодиться в будущем. Несомненно, теоретический материал, изложенный в книге, очень важен, но без практического применения он не стоит и ломаного гроша. На самом деле на подготовку файлов примеров понадобилось намного больше времени, чем на написание текста книги.

Файлы примеров на прилагаемом компакт-диске не заархивированы, поэтому можете открывать их, не копируя на жесткий диск компьютера.

Перекрестная ссылка

Дополнительные сведения о файлах, находящихся на прилагаемом компакт-диске, приведены в приложении Г.

О пакете Power Utility Pak

Популярный программный пакет Power Utility Pak — это непревзойденная коллекция полезных утилит Excel, а также новых функций рабочих листов. Он был разработан мною исключительно для совместного использования с VBA.

Я надеюсь, что этот программный продукт будет весьма полезен в вашей повседневной работе с Excel. Можете также приобрести полный набор исходных кодов VBA за номинальную плату. Изучив код, вы сможете овладеть рядом полезных методик программирования.

Пробную 30-дневную версию пакета Power Utility Pak можно загрузить на сайте `http://spreadsheetpage.com`.

Как пользоваться этой книгой

Вы можете работать с книгой, как вам удобно. Ваше решение прочесть ее от корки до корки я могу лишь поприветствовать. Но так как материал книги достаточно сложен, то порядок чтения глав часто не имеет значения. Подозреваю, что большинство читателей будут "перескакивать" от одной части к другой, выискивая то тут, то там полезные "лакомые кусочки". Если же вы столкнулись с трудностями, то сначала загляните в оглавление — нет ли в книге решения именно вашей проблемы.

Как связаться с автором

Мой издатель и я очень любим получать письма. Если вы являетесь счастливым обладателем этой книги, выделите несколько минут в своем напряженном рабочем графике для посещения сайта Wiley Publishing. И не просто посетите его, а оставьте на нем свои комментарии. (Перейдите по адресу `www.wiley.com` и щелкните на ссылке Contact Us.) Пожалуйста, будьте объективны в процессе оценивания. Если вы считаете, что в главе отсутствует нужная информация, дайте нам об этом знать. Конечно, как и все мы, я не слишком люблю получать отрицательные отзывы, но все же лучше "горькая правда".

От читателей моих книг я каждый день получаю по электронной почте не менее десяти вопросов. Благодарю вас за сотрудничество! К сожалению, у меня просто нет времени отвечать на них. В приложении А приведен полный список источников, которые помогут вам в решении той или иной проблемы.

Можете также посетить мой сайт, на котором найдете много разнообразной информации об Excel:

`http://spreadsheetpage.com`.

Ждем ваших отзывов!

Вы, читатель этой книги, и есть главный ее критик. Мы ценим ваше мнение и хотим знать, что было сделано нами правильно, что можно было сделать лучше и что еще вы хотели бы увидеть изданным нами. Нам интересны любые ваши замечания в наш адрес.

Мы ждем ваших комментариев и надеемся на них. Вы можете прислать нам бумажное или электронное письмо либо просто посетить наш веб-сервер и оставить свои замечания там. Одним словом, любым удобным для вас способом дайте нам знать, нравится ли вам эта книга, а также выскажите свое мнение о том, как сделать наши книги более интересными для вас.

Отправляя письмо или сообщение, не забудьте указать название книги и ее авторов, а также свой обратный адрес. Мы внимательно ознакомимся с вашим мнением и обязательно учтем его при отборе и подготовке к изданию новых книг.

Наши электронные адреса:

E-mail: `info@dialektika.com`
WWW: `http://www.dialektika.com`

Наши почтовые адреса:

в России: 127055, г. Москва, ул. Лесная, д. 43, стр. 1
в Украине: 03150, Киев, а/я 152

Часть I

Введение в Excel

В этой части...

Глава 1

Excel 2010: история программы

В этой главе...

- Краткая история электронных таблиц
- Почему программа Excel так удобна разработчикам
- Место Excel в стратегии Microsoft

Краткая история электронных таблиц

Многие склонны воспринимать электронные таблицы как нечто само собой разумеющееся. На самом же деле, хотя это, возможно, и трудно осознать, были времена, когда таких таблиц не было. Тогда вместо них люди использовали громоздкие ЭВМ или калькуляторы и тратили часы на работу, которую сейчас выполняют за минуту.

Все начиналось с VisiCalc

Первый в мире процессор электронных таблиц — программа VisiCalc — был создан Дэном Бриклином и Бобом Фрэнкстоном в 1978 году, когда в офисах еще даже не слышали о персональных компьютерах. VisiCalc была написана для компьютера Apple II — интересного маленького компьютера, игрушки по нынешним меркам. (Правда, в свое время Apple II днями напролет держал меня в состоянии гипноза.) VisiCalc в целом стала основой будущих электронных таблиц, а ее структуру строк и столбцов (а также синтаксис формул) до сих пор можно увидеть в современных электронных таблицах. VisiCalc быстро стала востребованной, и многие дальновидные компании приобретали Apple II лишь для того, чтобы создавать свои бюджетные планы с ее помощью. Со временем программе VisiCalc часто ставили в заслугу то, что именно она обеспечила компьютерам Apple II большую часть их первоначального успеха.

Тем временем появился новый вид персональных компьютеров, в которых использовалась операционная система CP/M. Компания Sorcim разработала SuperCalc — процессор электронных таблиц, который также привлек многих последователей.

И когда в 1981 году на сцене появился компьютер IBM PC, который вывел персональные компьютеры на массовый рынок, компания VisiCorp не стала медлить с переносом VisiCalc в новую аппаратную среду. Вскоре за ней последовала и Sorcim с версией SuperCalc, специально созданной для ПК.

По нынешним стандартам и VisiCalc, и SuperCalc — чрезвычайно незрелые программы. Например, текст, вводимый в ячейку, не должен был выходить за ее пределы, т.е. длинный заголовок приходилось вводить в несколько ячеек. Но как бы там ни было, возможность автоматизировать бюджетную рутину была по достоинству оценена — бумажным кассовым книгам тысячи бухгалтеров предпочли гибкие диски.

Совет

Копию оригинальной программы VisiCalc, созданной Дэном Бриклином, можно загрузить с его веб-сайта (`www.bricklin.com`). Даже 30 лет спустя после разработки этой программы (размером всего лишь 27 Кбайт) ее можно использовать на современных компьютерах (рис. 1.1).

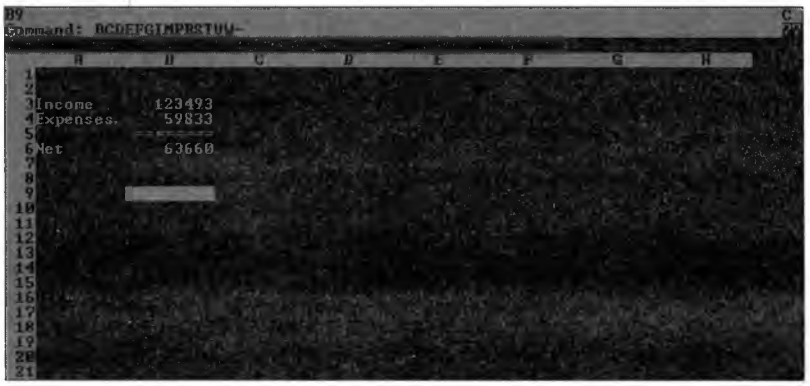


Рис. 1.1. Программа VisiCalc, выполняемая в среде DOS на компьютере под управлением Windows XP

Lotus 1-2-3

Оценив успех VisiCalc, небольшая группа компьютерных гениев из компании, только что основанной в Кембридже, штат Массачусетс, решила усовершенствовать концепцию электронных таблиц. Под руководством Митча Капора и Джонатана Сакса эта организация разработала новый продукт и провела первую в компьютерной отрасли законченную маркетинговую подготовку. Я помню, как рассматривал в *The Wall Street Journal* рекламу Lotus 1-2-3, напечатанную крупным шрифтом. Это был первый на моей памяти случай, когда в прессе появилась реклама программного продукта.

Выпущенный в январе 1983 года компанией Lotus Development Corporation, этот продукт имел мгновенный успех. Несмотря на этикетку с ценой 495 долларов (да, люди действительно платили столько за программный продукт), что эквивалентно современной тысяче долларов, он по степени популярности быстро обогнал VisiCalc, взлетел на вершину продаж и остался там на многие годы.

Причины успеха Lotus 1-2-3

Процессор электронных таблиц Lotus 1-2-3 не только превосходил VisiCalc и SuperCalc по всем основным параметрам, но и стал первой программой, использовавшей новые уникальные возможности мощной 16-разрядной архитектуры IBM PC AT. Например, программа Lotus 1-2-3 игнорировала медленные вызовы DOS и передавала данные непосредственно в видеопамять, производя впечатление невероятной производительности, довольно необычной на то время. Прорывом была также интерактивная справочная система, а хитроумные "движущиеся" панели меню стали стандартом на многие годы.

Впрочем, существовала возможность, которая действительно выделяла Lotus 1-2-3 среди остальных процессоров электронных таблиц. Речь идет о средстве создания *макросов* — поистине мощном инструменте, который позволял пользователям электронных таблиц записывать выполняемые ими операции и таким образом автоматизировать многие процессы. Когда выполнялся указанный макрос, записанные в нем операции передавались в приложение. И хотя ему далеко до нынешних инструментов записи, в Lotus 1-2-3 был заложен фундамент будущих технологий, интегрируемых во все процессоры электронных таблиц.

Lotus 1-2-3 — первый (и к тому же успешный) интегрированный пакет, в котором система производительных электронных таблиц (1) сочеталась с элементарной графикой (2) и ограниченными, но невероятно удобными средствами управления базами данных (3). Теперь понятно, почему "1-2-3"?

Компания Lotus постаралась, чтобы вслед за первым выпуском пакета Lotus 1-2-3 в апреле 1983 года последовал выпуск 1A. Этот новый программный продукт имел огромный успех и предоставил Lotus завидное положение монополиста на рынке процессоров электронных таблиц. В сентябре 1985 года выпуск 1A был заменен выпуском 2, а в июле следующего года — выпуском 2.01 с исправленными ошибками. Выпуск 2, в отличие от предыдущих, поддерживал *надстройки* (add-ins) — специальные программы, которые можно интегрировать в приложение, чтобы расширить его возможности. Кроме того, в выпуске 2 содержалась усовершенствованная система управления памятью, предлагалось больше функций, а максимальное количество строк увеличилось в четыре раза по сравнению с предыдущими версиями. В данной версии также поддерживался математический сопроцессор и содержался усовершенствованный макроязык, популярность которого превысила самые смелые мечты его разработчиков.

Не удивительно, что успех Lotus 1-2-3 способствовал появлению *клонов* — похожих программных продуктов, в которых обычно предлагалось несколько дополнительных возможностей и которые, как правило, продавались намного дешевле. Среди более или менее заметных стоит упомянуть Twin компании Mosaic Software и серию VP Planner компании Paperback Software. В конце концов, за нарушение авторских прав (копирование внешнего вида Lotus 1-2-3) компания Lotus возбудила против Paperback Software судебное дело. Исход этого дела, успешный для Lotus, по сути, привел к банкротству Paperback.

Летом 1989 года Lotus выпустила DOS- и OS/2-варианты долгожданной версии 3. К электронным таблицам, состоящим из уже ставших привычными строк и столбцов, этот продукт добавил новое измерение. Такое "расширение парадигмы" было достигнуто путем увеличения количества страниц в электронных таблицах. Впрочем, новой данная мысль в действительности не была. Идея трехмерных электронных таблиц впервые применялась в относительно малоизвестном продукте Boeing Calc; ее также реализовали в таких программах, как SuperCalc 5 и CubeCalc.

В версии 3 пакета Lotus 1-2-3 содержались многие полезные пользовательские инструменты, которые в конце концов стали стандартными. Речь идет о многоуровневых электронных таблицах, одновременной работе с большим количеством файлов, их связывании, усовершенствованной графике и прямом доступе к внешним файлам баз данных. Однако в этой версии отсутствовала важная возможность, о которой мечтали многие пользователи: не был реализован высококачественный вывод информации.

Версия 3 начала свой путь с малого рыночного потенциала, поскольку требовала для нормальной работы компьютер на базе процессора 80286 с минимальным объемом оперативной памяти 1 Мбайт — требования довольно "жесткие" для 1989 года. И тут Lotus вытащила туз, припрятанный в ее корпоративном рукаве. Одновременно с объявлением о появлении версии 3 компания удивила буквально всех, заявив об усовершенствовании версии 2.01 (усовершенствованный продукт материализовался через несколько месяцев в виде Lotus 1-2-3 версии 2.2). Вопреки ожиданию большинства аналитиков версия 3 не заменила версию 2. Вместо этого компания Lotus сделала блестящий ход, разбив рынок процессоров электронных таблиц на два сегмента: тот, который работает на высокопроизводительном оборудовании, и тот, для которого по карману более скромный компьютер.

Слишком мало, слишком поздно...

Конечно, для фанатов электронных таблиц версия 2.2 продукта Lotus 1-2-3 панацеей не стала, но все-таки она значительно расширила возможности пользователей. Самой важной возможностью этой версии была надстройка *Allways*, которая позволяла "творить" привлекательные отчеты, выполненные с использованием разнообразных шрифтов, обрамлений и заливок. Кроме того, просмотр полученных результатов на экране выполнялся в режиме WYSIWYG (What You See Is What You Get — что видишь, то и получаешь). Впрочем, когда пользователи просматривали и редактировали свою работу в этом режиме, они не могли управлять данными электронных таблиц. Но, несмотря на такое суровое ограничение, большинство пользователей Lotus 1-2-3 было вне себя от радости, потому что, имея в арсенале такую новую возможность, они наконец-то смогли создавать документы почти типографского качества.

В мае 1990 года компания Microsoft выпустила Windows 3.0. Как вы, возможно, знаете, эта система привела к изменению принципов использования персонального компьютера. Видимо, специалисты, принимавшие в Lotus решения, не считали Windows серьезным продуктом, и компания не спешила презентовать свою первую программу, работающую с электронными таблицами в Windows. Такая программа — Lotus 1-2-3 для Windows — была выпущена только в конце 1991 года. Хуже того, этот продукт, если судить объективно, оказался неудачным. Он не смог использовать преимущества среды Windows и разочаровал многих пользователей. Эта программа также разочаровала по крайне мере одного автора компьютерных книг. Моя первая книга называлась *PC World 1-2-3 For Windows Complete Handbook*, причем продать удалось менее тысячи экземпляров.

В результате программа Excel, которая уже заявила о себе как о "главном" в Windows процессоре электронных таблиц, стала единоличным лидером на рынке подобных Windows-программ (и с тех пор никогда не сдавала этой позиции). Что касается Lotus, то в июне 1993 года вышла очередная ее версия: Lotus 1-2-3 версии 4 для Windows. Она была значительно лучше своего оригинала. Версия 5 этой программы для Windows появилась в середине 1994 года.

В то же время Lotus выпустила версию 4.0 этого продукта для DOS (Lotus 1-2-3 Release 4.0 for DOS). Многие аналитики (и я в том числе) ожидали появления продукта,

более совместимого с Windows. Однако мы ошиблись; эта версия стала лишь более усовершенствованной по сравнению с версией 3.4. Поскольку операционная система Windows получила невероятное распространение, это оказалась последняя версия Lotus 1-2-3 для DOS, которая увидела свет.

Со временем электронные таблицы стали для Lotus менее важными (ее ведущим продуктом стал Notes). В середине 1995 года компания IBM приобрела Lotus Development Corporation. Появились еще две версии Lotus 1-2-3, но это, как говорится, был тот случай, когда "и слишком мало, и слишком поздно". Excel явно доминирует на рынке процессоров электронных таблиц, а Lotus 1-2-3 продолжает терять свои позиции.

Quattro Pro

Еще одним заслуживающим внимания игроком в сфере электронных таблиц является (или, лучше сказать, являлась) компания Borland International. В 1994 году компания Novell купила у WordPerfect International и у Borland весь их бизнес, связанный с процессорами электронных таблиц. А в 1996 году и WordPerfect, и Quattro Pro были выкуплены компанией Corel Corporation. На ниве электронных таблиц компания Borland начала работать в 1987 году, выпустив продукт, называемый *Quattro*. Внутри компании код этой программы назывался *Buddha*, поскольку эта программа позиционировалась как достойный конкурент Lotus (лидера рынка). Это, по сути, был клон Lotus 1-2-3, который имел несколько дополнительных средств и, возможно, более хорошую систему меню. Кроме того, указанный продукт был значительно дешевле. Важно еще и то, что пользователи могли выбрать систему меню, похожую на применяемую в Lotus 1-2-3, и, таким образом, использовать знакомые команды, а также обеспечивать совместимость с макросами Lotus 1-2-3.

Осенью 1989 года компания Borland начала продавать Quattro Pro — более мощный продукт, созданный на базе, отличной от исходной Quattro, и превосходивший Lotus 1-2-3 буквально во всех аспектах. Например, первая версия Quattro Pro позволяла работать с большим количеством рабочих листов, находящихся в окнах, которые можно было перемещать и размеры которых можно было менять. Даже при том, что у программы не было графического пользовательского интерфейса. Еще одна деталь: Quattro Pro создавалась на основе малоизвестного продукта Surpass, приобретенного Borland.

В конце 1990 года была выпущена версия 2.0 программы Quattro Pro (Quattro Pro Version 2.0), которая уже имела поддержку трехмерной графики и обеспечивала связь с базами данных Paradox от Borland. Всего лишь полгода спустя — к большому огорчению конкурентов — появилась версия Quattro Pro 3.0, в которой по желанию можно было настраивать графический пользовательский интерфейс и допускался просмотр данных в режиме слайд-шоу. Весной 1992 года появилась версия 4, в которой предлагались настраиваемые "быстрые" панели, а также новая возможность — применение аналитической графики. Что касается версии 5, вышедшей в 1994 году, она характеризовалась единственным новшеством, которое можно назвать серьезным, — наличием блокнотов рабочих листов (т.е. трехмерных рабочих листов).

Как и Lotus, компания Borland не спешила переходить на сторону Windows. Впрочем, когда осенью 1992 года Quattro Pro для Windows поступила в продажу, она составила довольно серьезную конкуренцию двум другим Windows-программам, работавшим с электронными таблицами: Excel 4.0 и Lotus 1-2-3 версии 1.1 для Windows. Важно то, что в Quattro Pro для Windows предлагалась новая возможность, известная как UI Builder

(построитель пользовательского интерфейса). Она позволяла разработчикам и опытным пользователям легко создавать индивидуальные варианты пользовательского интерфейса.

Кроме того, ни к чему не привела судебная тяжба между Lotus и Borland. Вначале Lotus ее выиграла, заставив Borland удалить из Quattro Pro поддержку макросов Lotus 1-2-3 и возможность создания таких же меню, как и в Lotus 1-2-3. Однако со временем, в конце 1994 года, это решение было пересмотрено, и теперь в Quattro Pro в полной мере поддерживаются средства, обеспечивающие совместимость с Lotus 1-2-3 (как будто они действительно кому-то нужны). На эту долгую борьбу обе стороны потратили миллионы долларов, а когда пыль улеглась, то настоящего победителя так и не оказалось.

Позднее Borland выпустила оригинальную версию 5 программы Quattro Pro для Windows. После того как компания Novell получила от Borland все, что касалось процессоров электронных таблиц, версия 5 была модернизирована до версии 6.

В 1996 компания Corel Corporation приобрела продукты WordPerfect и Quattro Pro. На момент написания этой книги выпущена версия Quattro Pro 14, которая входит в состав пакета WordPerfect Office X4.

Для разработчиков электронных таблиц пакет Quattro Pro долгое время был пределом совершенства, но затем появилась программа Excel 5.

Microsoft Excel

А теперь пришло время поближе познакомиться с героем нашего романа.

Многие читатели не знают, что по части электронных таблиц компания Microsoft стала приобретать опыт еще в начале 1980-х годов. И за эти годы соответствующие программы Microsoft прошли долгий путь развития: все началось с MultiPlan, отвечавшей лишь минимальным требованиям, и закончилось Excel 2010, представляющей последние разработки в этой области.

Сначала MultiPlan

В 1982 году Microsoft выпустила программу MultiPlan — свой первый продукт для работы с электронными таблицами. Предназначенная для компьютеров, которые работают под управлением операционной системы CP/M, программа вскоре была перенесена на другие платформы, в том числе на Apple II, Apple III, XENIX и MS-DOS.

MultiPlan преимущественно игнорировала стандарты пользовательского интерфейса для программ. Сложная для изучения и применения, эта программа так никогда и не приобрела в США особой популярности. И не удивительно, что ее достаточно быстро обогнала Lotus 1-2-3.

Пришествие Excel

От MultiPlan берет свое начало программа Excel, впервые зарекомендовавшая себя на платформе Macintosh в 1985 году. Как и все Mac-приложения, Excel являлась графической программой (в отличие от текстовой MultiPlan). В ноябре 1987 года компания Microsoft выпустила первую версию Excel, предназначенную для Windows (она была названа Excel 2.0 для Windows, чтобы сохранить преемственность с номером версии, выпущенной для Macintosh). Поскольку тогда операционная система Windows не имела широкого распространения, то в состав Excel 2.0 вошла исполняемая версия Windows, предназначенная исключительно для обеспечения работы Excel. Менее чем через год Microsoft выпустила новую версию Excel — 2.1 (Excel Version 2.1). В июле 1990 года компания предложила небольшое обновление (2.1b), совместимое с Windows 3.0.

И хотя версии 2.*x* были по современным меркам довольно ограниченными (рис. 1.2) и не имели эффектного внешнего вида последних версий, они все равно привлекли пусть и небольшую, но верную группу поддержки и заложили прекрасный фундамент для будущих разработок.

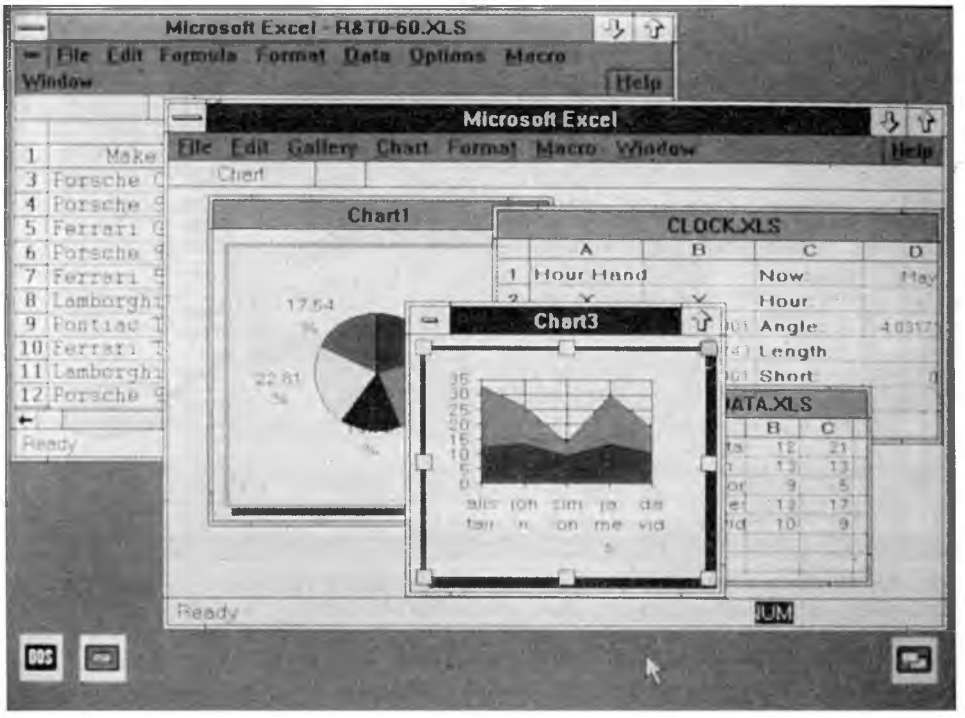
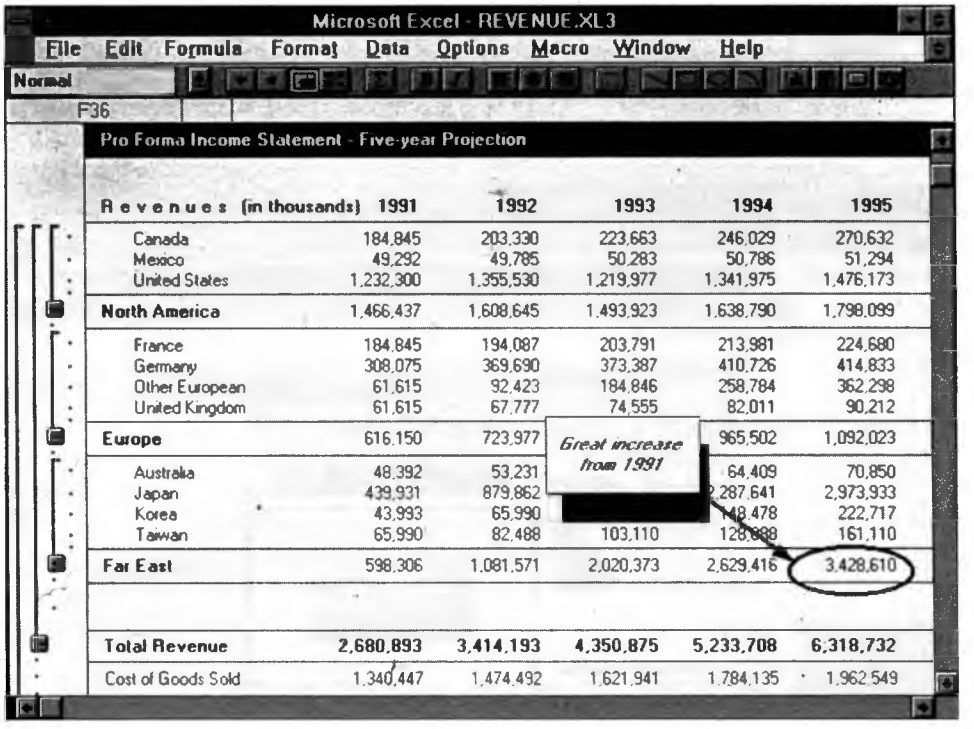


Рис. 1.2. Оригинальная программа Excel 2.1 для Windows *(фотография предоставлена компанией Microsoft)*

Программа Excel имела встроенный макроязык (XLM), который состоял из функций, обрабатываемых одна за другой. Этот макроязык был достаточно мощным, но сложным для изучения и применения. Как вы увидите, на смену XML пришел VBA, которому и посвящена настоящая книга. Но в то же время даже Excel 2010 поддерживает макросы XLM.

Кроме того, компания Microsoft разработала версию Excel (под номером 2.20) для OS/2 Presentation Manager. Она была выпущена в сентябре 1989 года, и примерно десять месяцев спустя появилось ее обновление (версия 2.21). Впрочем, несмотря на усилия со стороны IBM, операционная система OS/2 никогда не пользовалась особой популярностью.

В декабре 1990 года Microsoft выпустила Excel 3 для Windows со значительными усовершенствованиями как внешнего вида, так и возможностей (рис. 1.3). Среди новинок были панель инструментов, средства рисования, мощный инструмент поиска решения, поддержка надстроек и технологии OLE (Object Linking and Embedding — связывание и внедрение объектов), трехмерные диаграммы, кнопки для макросов, упрощенная консолидация файлов, редактирование в составе рабочих групп и перенос по словам текста внутри ячейки. Кроме того, в Excel 3 существовала возможность работать с внешними базами данных (с помощью программы Q+E). Пять месяцев спустя появилось обновление Excel для OS/2.



Revenues (in thousands)	1991	1992	1993	1994	1995
Canada	184,845	203,330	223,663	246,029	270,632
Mexico	49,292	49,785	50,283	50,786	51,294
United States	1,232,300	1,355,530	1,219,977	1,341,975	1,476,173
North America	1,466,437	1,608,645	1,493,923	1,638,790	1,798,099
France	184,845	194,087	203,791	213,981	224,680
Germany	308,075	369,690	373,387	410,726	414,833
Other European	61,615	92,423	184,846	258,784	362,298
United Kingdom	61,615	67,777	74,555	82,011	90,212
Europe	616,150	723,977		965,502	1,092,023
Australia	48,392	53,231		64,409	70,850
Japan	439,931	879,862		2,287,641	2,973,933
Korea	43,993	65,990		148,478	222,717
Taiwan	65,990	82,488	103,110	128,888	161,110
Far East	598,306	1,081,571	2,020,373	2,629,416	3,428,610
Total Revenue	2,680,893	3,414,193	4,350,875	5,233,708	6,318,732
Cost of Goods Sold	1,340,447	1,474,492	1,621,941	1,784,135	1,962,549

Рис. 1.3. Программа Excel 3 была существенно улучшена по сравнению со своей предшественницей *(фотография предоставлена компанией Microsoft)*

Версию 4, выпущенную весной 1992 года, было не только легче использовать. Она являлась более мощной и содержала большее количество инструментов, предназначенных для опытных пользователей (рис. 1.4). Буквально в каждом обзоре компьютерных журналов, где сравнивались процессоры электронных таблиц, Excel 4 оказывалась на самом почетном месте. Тем временем отношения между Microsoft и IBM изменились к худшему; Excel 4 для операционной системы OS/2 так никогда и не была выпущена, а Microsoft прекратила выпуск версий Excel, предназначенных для этой системы.

Рождение VBA

Версия Excel 5 предстала перед публикой в начале 1994 года и сразу же получила восторженные отзывы. Как и ее предшественница, она занимала верхнюю строчку во всех рейтингах процессоров электронных таблиц, публиковавшихся ведущими коммерческими журналами. Несмотря на жесткую конкуренцию с Lotus 1-2-3 выпуска 5 для Windows и Quattro Pro для Windows — а ведь и тот, и другой продукты могли решить буквально любую задачу, которую подбрасывали им электронные таблицы, — Excel 5 все равно продолжала задавать тон. Кстати, эта версия была первой, в которой использовался язык VBA.

Версия Excel 95 (известная также как Excel 7) была выпущена одновременно с Microsoft Windows 95. Microsoft специально пропустила шестой номер, чтобы продукты, входящие в ее пакет Office, имели одинаковые номера версий. На первый взгляд, Excel 95 незначительно отличается от Excel 5. Однако существенная часть кода ее ядра была переписана, а во многих местах наблюдалось заметное увеличение быстродействия. Важно и то, что

в Excel 95 использовался тот же формат файлов, что и в Excel 5. Это был первый случай, когда усовершенствованной версии Excel не предоставили новый формат файла. Впрочем, совместимость не стала до конца полной, поскольку в языке VBA появились некоторые усовершенствования. Следовательно, с помощью Excel 95 можно было разрабатывать приложения, которые загружались в Excel 5 (хотя и не работали так, как положено).

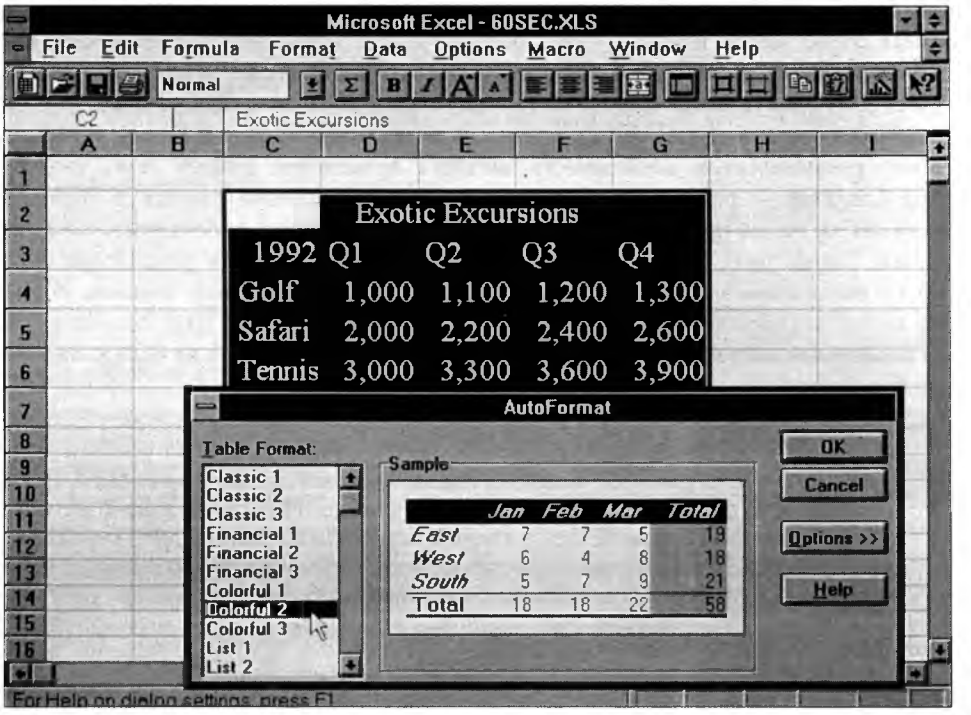


Рис. 1.4. Версия Excel 4 являлась еще одним шагом вперед, хотя до Excel 5 ей было далеко *(фотография предоставлена компанией Microsoft)*

В начале 1997 года Microsoft выпустила интегрированный пакет программ Office 97, в состав которого входила Excel 97 (Excel 97 еще называется Excel 8). Эта версия характеризовалась многими общими изменениями, а также абсолютно новым интерфейсом для разработки приложений на основе VBA. Был также предложен совершенно новый способ разработки пользовательских диалоговых окон (которые теперь назывались не диалоговыми листами, а пользовательскими формами). Microsoft попыталась сделать Excel 97 совместимой с предыдущими версиями, но эта совместимость оказалась далекой от совершенства. Чтобы многие приложения, разработанные с помощью Excel 5 или Excel 95, могли работать в Excel 97 или более поздних версиях, приходилось прибегать к определенным уловкам.

Перекрестная ссылка

Вопросы совместимости будут рассмотрены в главе 26.

Программа Excel 2000 была выпущена в начале 1999 года; она продавалась как часть интегрированного офисного пакета Office 2000. Усовершенствования, которые пред-

ставлены в Excel 2000, относятся в основном к работе в Интернете, хотя некоторые значительные изменения заметны и в области программирования.

Excel 2002 (известная также под названием Excel XP) появилась на рынке в середине 2001 года. Как и ее предшественница, новыми возможностями, которые можно назвать серьезными, эта программа не располагала. Впрочем, появились некоторые новшества и были внесены незначительные корректировки в уже имеющиеся возможности. Вероятно, самая существенная из них — это способность восстанавливать поврежденные файлы и сохранять работу пользователя при аварийном завершении Excel.

Excel 2003 (выпущенная осенью 2003 года) стала одним из самых разочаровывающих обновлений в линейке Excel. В ней оказалось очень мало новинок. Компания Microsoft усиленно рекламировала возможность импорта и экспорта файлов XML (eXtensible Markup Language — расширяемый язык разметки) и привязки данных к конкретным ячейкам листа, но, по правде говоря, все это требовалось очень небольшому числу пользователей. Кроме того, Microsoft немного расширила возможности управления правами доступа к различным элементам рабочей книги (позволив, например, задавать пользователей, которым разрешен просмотр конкретных листов). В Excel 2003 появилась также новая справочная система (содержимое справки выводилось теперь на панель задач), которую дополнило новое средство поиска на панели задач.

Примечание

По определенным причинам компания Microsoft решила продавать две подверсии Excel 2003. Поддержка XML и прав доступа реализована в отдельно продаваемой версии Excel, а также в версии Excel, входящей в состав пакета Office 2003 Professional. В результате разработчики Excel вынуждены были решать проблемы совместимости в пределах одной версии.

Новый пользовательский интерфейс

В конце 2006 года появилась программа Excel 2007 (версия 12), которая входила в пакет Microsoft Office 2007. В этой версии процессора электронных таблиц пользовательский интерфейс претерпел радикальные изменения. На смену прежним меню и панелям инструментов пришел пользовательский интерфейс в виде ленты. Размер сетки, состоящей из ячеек, в Excel 2007 в тысячу раз больше, чем размер сетки в предыдущих версиях, а для рабочих файлов предложен новый формат Open XML. Другие изменения коснулись таблиц и условного форматирования, многочисленным косметическим улучшениям подверглись диаграммы и темы документов.

Реакция пользователей на появление нового интерфейса была весьма противоречивой. Одни горячо его полюбили, другие же, наоборот, возненавидели. Некоторые компании даже разработали специальные надстройки, с помощью которых пользователи Excel 2007 могут вернуться к привычным старым меню. Конечно, для начинающих программа Excel 2007 проще, но опытные пользователи потратят массу времени на поиск своих любимых команд.

Программа Excel 2010, рассматриваемая в этой книге, является частью пакета Microsoft Office 2010. И что интересно, разработчики из Microsoft перестраховались. Порядковый номер версии новой программы должен был стать 13, но сработал эффект страха перед числом 13, и версия новой программы Excel получила номер 14.

В Excel 2010 существенно расширились возможности сводных таблиц, условного форматирования и коррекции изображений. Появились компактные диаграммы, занимающие одну-две ячейки, которые получили название *спарклайны*. В процессе вставки данных теперь можно предварительно увидеть, что из всего этого получится. Новое

представление Backstage включает ряд команд, рассчитанных на работу с документами, например сохранение и печать. Также пользователи Excel получили возможность настраивать ленту. И наконец, в распоряжении пользователей оказались десятки новых функций рабочего листа, большинство из которых относятся к категории высокоспециализированных, пришедших на смену прежним функциям, при использовании которых возникали определенные проблемы с точностью вычислений.

Современный рынок электронных таблиц

Итак, более чем 30-летняя история разработки электронных таблиц уместилась всего лишь на нескольких страницах. Это был достаточно интересный обзор, и мне приятно, что я был свидетелем всех этих процессов.

Времена меняются. Сегодня Microsoft не просто доминирует на рынке электронных таблиц, она полностью его контролирует. Небольшая конкуренция имеет место в нише программ "с открытым исходным кодом", например OpenOffice и StarOffice. Наверное, вы слышали о многообещающих электронных таблицах в Интернете, таких как Google Spreadsheets (рис. 1.5). В ответ на это Microsoft предложила пользователям собственную веб-версию Excel и других приложений Office 2010.

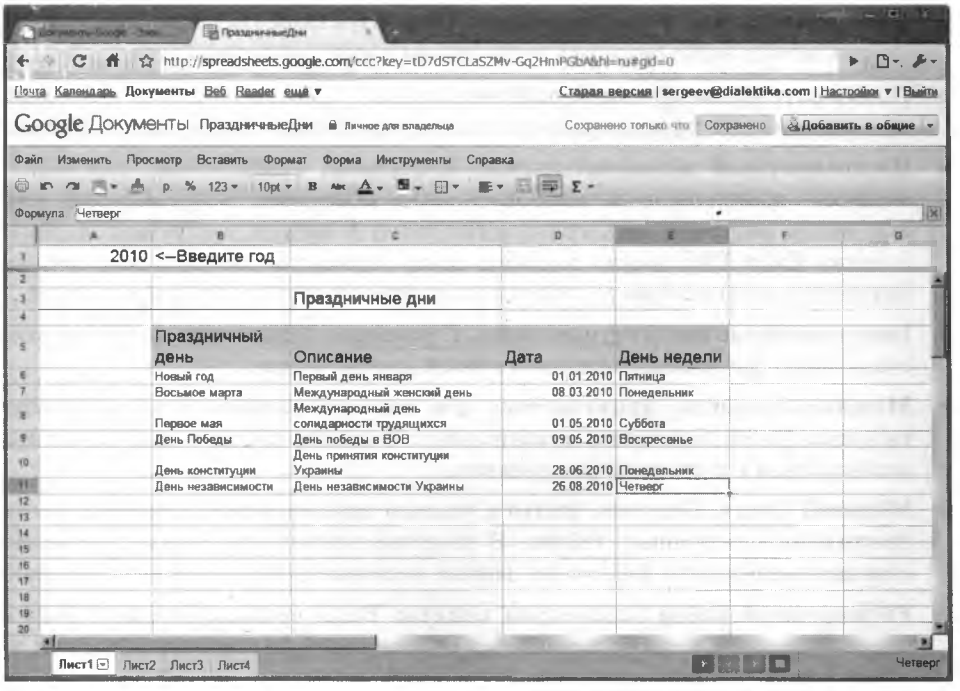


Рис. 1.5. Веб-версия электронной таблицы от Google

На самом деле подобные конкуренты не представляют серьезной угрозы для Microsoft. Фактически самый серьезный конкурент для компании Microsoft — она сама. Пользователи предпочитают работать с определенной версией Excel и очень редко переходят на новую версию. Заставить их перейти на совершенно новую версию Excel — наибольший вызов для компании Microsoft.

Почему программа Excel так удобна разработчикам

Программа Excel обладает исключительной гибкостью с точки зрения программирования и является наилучшим выбором для разработки приложений электронных таблиц.

Ниже перечислены ключевые свойства Excel с точки зрения разработчика.

- **Файловая структура.** Благодаря "многолистовой" структуре облегчается организация элементов приложения и их хранение в отдельном файле. Например, в состав единственного файла рабочей книги может входить произвольное число рабочих листов и листов диаграмм. Здесь же хранятся пользовательские формы и модули VBA, которые невидимы для конечного пользователя.
- **Visual Basic for Applications.** С помощью этого макроязыка можно создавать структурированные программы непосредственно в Excel. Именно использованию этого языка, представляющего собой сочетание мощи и простоты, посвящена данная книга.
- **Упрощенный доступ к элементам управления.** В Excel до предела упрощено добавление новых элементов управления на рабочий лист, таких как кнопки, списки и кнопки выбора. При этом создавать макросы зачастую вообще не требуется.
- **Пользовательские диалоговые окна.** Используя такое средство, как окна `UserForms`, можно легко создавать профессиональные диалоговые окна.
- **Пользовательские функции рабочего листа.** С помощью VBA можно создавать пользовательские функции рабочего листа, упрощающие формулы и вычисления.
- **Настраиваемый пользовательский интерфейс.** Разработчики получили возможность контролировать пользовательский интерфейс. В предыдущих версиях они могли создавать пользовательские меню и панели инструментов. Начиная с версии Excel 2007 можно изменять ленту. Изменить внешний вид ленточного интерфейса не столь просто, как в предыдущих версиях, но вполне возможно.
- **Настраиваемые контекстные меню.** С помощью VBA можно настроить контекстные меню, вызываемые щелчком правой кнопки мыши.
- **Мощные команды, применяемые для анализа данных.** Благодаря сводным таблицам Excel облегчается суммирование больших объемов данных, не требуя особых усилий со стороны пользователя.
- **Microsoft Query.** Получить доступ к важным данным можно непосредственно из программы электронных таблиц. В качестве источников данных применяются стандартные форматы файлов баз данных, тестовые файлы, а также веб-страницы.
- **Расширенный набор параметров защиты.** Ваши приложения могут быть конфиденциальными, а также защищаться от изменений со стороны случайных пользователей.
- **Возможность создания подключаемых модулей.** Выполнив одну-единственную команду, можно создать файлы подключаемых модулей, которые добавляют новые возможности в Excel.
- **Поддержка автоматизации.** С помощью VBA можно контролировать другие приложения, которые поддерживают автоматизацию. Например, VBA-макрос может генерировать отчет в Microsoft Word.

- **Возможность создания веб-страниц.** HTML-документ можно создать на основе рабочей книги Excel. Конечно, HTML-код станет чрезмерно "раздутым", но зато будет восприниматься веб-браузерами.

Место Excel в стратегии Microsoft

В настоящее время большинство копий Excel продаются в составе Microsoft Office — прикладного пакета, включающего множество других программ (состав пакета зависит от версии Office). Это удобно, поскольку облегчает взаимодействие программ, и именно эту тенденцию поощряет компания Microsoft. Все приложения Office имеют похожий интерфейс пользователя, и все они поддерживают VBA.

Таким образом, приобретя навыки программирования на VBA в Excel, вы сможете применить их и в других приложениях — понадобится лишь изучить объектную модель для этих приложений.

Глава 2

Основные элементы Excel

В этой главе...

- Объектное мышление
- Рабочие книги
- Пользовательский интерфейс Excel
- Настройка окна программы
- Ввод данных
- Формулы, функции и имена
- Выделение объектов
- Форматирование
- Параметры защиты
- Диаграммы
- Фигуры и рисунки SmartArt
- Доступ к базам данных
- Excel и Интернет
- Инструменты анализа
- Надстройки
- Макросы и программирование
- Файловые форматы
- Справочная система Excel

Эта глава посвящена рассмотрению основных понятий, связанных с Excel. Здесь же рассматриваются новые возможности, которые появились в Excel 2010.

Объектное мышление

Планируя разработку приложений с помощью Excel (особенно с помощью VBA), следует проанализировать концепцию *объектов* — элементов Excel, которыми можно манипулировать напрямую или с помощью макросов. Ниже приведены примеры объектов в Excel:

- само приложение Excel;
- рабочая книга Excel;
- лист в рабочей книге;
- диапазон или таблица в рабочем листе;
- элемент управления `ListBox` (список) в пользовательской форме (пользовательское диалоговое окно);
- диаграмма на листе диаграммы;
- ряд данных на диаграмме;
- отдельная точка данных на диаграмме.

Обратите внимание, что в приведенном списке соблюдается *иерархия объектов*: объект Excel содержит объекты рабочих книг, в которых находятся объекты рабочих листов, а те, в свою очередь, включают объекты диапазонов ячеек. Подобная иерархия составляет *объектную модель* Excel. В Excel насчитывается около двухсот классов объектов, и этими объектами можно управлять непосредственно или с помощью VBA. Собственные объектные модели имеют и другие программные продукты Microsoft Office.

Примечание

При разработке приложений один из наиболее важных моментов — контроль над объектами. На протяжении книги рассматриваются способы автоматизации контроля над объектами Excel, в том числе с помощью VBA. Все это подробнее рассматривается в следующих главах.

Рабочие книги

Среди объектов Excel самым распространенным является *рабочая книга*. Все, что вы делаете в Excel, происходит в рабочей книге, которая хранится в файле, имеющем по умолчанию расширение XLSX. В рабочей книге Excel может содержаться любое количество листов (предел зависит от объема оперативной памяти). Все они делятся на четыре вида:

- рабочие листы;
- листы диаграмм;
- листы макросов XLM (устаревшие, но до сих пор поддерживаемые);
- диалоговые листы (также устаревшие, но до сих пор поддерживаемые).

Можно открыть любое количество рабочих книг (каждую в своем окне), но в любой момент только одна из них может быть активной. Аналогично, *активным листом* может быть только один из листов рабочей книги. Чтобы активизировать лист, щелкните на его ярлыке, расположенном в нижней части экрана. Для изменения имени листа дважды щелкните на ярлыке и введите новое название. Если на ярлыке щелкнуть правой кнопкой мыши, то появится контекстное меню, команды которого позволяют изменить цвет ярлыка, скрыть лист и т.д.

Для сокрытия окна рабочей книги воспользуйтесь командой Вид⇨Окно⇨Скрыть окно (View⇨Window⇨Hide). Скрытая рабочая книга остается открытой, но невидимой для пользователя. Для отображения скрытой рабочей книги воспользуйтесь командой Вид⇨Окно⇨Отобразить окно (View⇨Window⇨Unhide). Одна и та же рабочая книга может отображаться в различных окнах (для этого выберите команду Вид⇨Окно⇨Новое окно (View⇨Window⇨New Window)). В окнах могут отображаться разные рабочие листы либо различные области одного и того же листа.

Рабочие листы

Самыми распространенными объектами являются рабочие листы. Говоря об электронной таблице, пользователи обычно подразумевают рабочий лист. Он состоит из ячеек, которые содержат данные и формулы.

Каждый рабочий лист в Excel 2010 состоит из 16 384 столбцов и 1 048 576 строк. Можно временно скрыть ненужные строки и столбцы, но вам не удастся превысить предел, ограничивающий максимальное число строк и столбцов.

Примечание

В версиях, предшествующих Excel 2007, использовался двоичный файловый формат XLS, и рабочие листы содержали до 65 536 строк и 256 столбцов. При открытии подобного файла Excel 2010 переходит в *режим совместимости*, в котором выбирается меньший размер сетки рабочего листа. Для преобразования подобного файла в новый формат сохраните его в виде XLSX- или XLSM-файла. Затем закройте рабочую книгу и повторно откройте ее.

Насколько велик ваш рабочий лист

Иногда полезно оценить физические размеры рабочего листа. Умножив количество столбцов на количество строк (16384×1048576), получим величину, равную 17 179 869 184 ячейкам. Причем это количество относится только к одному рабочему листу. А в рабочей книге содержится несколько рабочих листов.

Если выбрано разрешение экрана 1600×1200 пикселей, а высота строк и ширина столбцов оставлены заданными по умолчанию, на экране одновременно смогут отображаться 24 столбца и 49 строк (либо 1176 ячеек) — примерно 0,000068% от общего числа ячеек рабочего листа. Другими словами, информация, находящаяся на одном рабочем листе, займет более 14,6 млн. экранов.

Если в каждую ячейку рабочего листа введена одна цифра и пользователь просматривает клип со скоростью одна ячейка в секунду, у него уйдет более 500 лет на просмотр всех ячеек рабочего листа. Для вывода на печать всех просмотренных ячеек понадобится более 36 млн. листов бумаги, которые образуют стопку высотой около 4 км.

Нетрудно предположить, что заполнить всю книгу значениями просто невозможно. Нереально даже приблизиться к этому. Даже если вы используете 64-разрядную версию Excel, размер свободной памяти исчерпается моментально, а Excel завершит свое выполнение отображением сообщения об ошибке.

Предоставляемая возможность применения в рабочей книге большого количества рабочих листов ценна даже не тем, что вы получаете доступ к большему числу ячеек. Преимущество заключается в другом — большое количество рабочих листов позволяет лучше организовать документ. Ранее, когда файл содержал только один рабочий лист, разработчики теряли немало времени, пытаясь организовать рабочий лист так, чтобы

информация в нем хранилась наиболее рационально. Теперь можно хранить ее в любом количестве рабочих листов и все равно иметь к ней мгновенный доступ (для этого потребуется щелкнуть на ярлыке нужного листа).

Примечание

Создаваемые рабочие книги по умолчанию состоят из трех рабочих листов. На практике чаще всего требуется один рабочий лист, остальные же легко добавляются в случае необходимости. Для изменения количества рабочих листов, заданного по умолчанию, выполните команду Файл⇨Параметры Excel (File⇨Excel Options), выберите вкладку Общие (General) и задайте соответствующее значение в списке Число листов (Include This Many Sheets).

В ячейке рабочего листа находится постоянное значение или результат выполнения формулы. В качестве значения может использоваться число, дата, булево значение ("истина" или "ложь") или текст. Кроме того, каждый рабочий лист имеет скрытый графический слой, который позволяет вставлять графические объекты (такие, например, как диаграммы, фигуры, рисунки SmartArt, элементы управления пользовательских форм, рисунки и встроенные объекты).

Можно полностью контролировать ширину столбцов и высоту строк — есть даже возможность скрывать строки и столбцы (так же, как и целые рабочие листы). Текст внутри ячейки может отображаться вертикально (или под углом) и даже переноситься по словам, занимая в пределах ячейки более одной строки.

Примечание

Ранее палитра цветов в Excel имела размер 56 ячеек. Начиная с Excel 2007 количество цветов практически не ограничено. Кроме того, в Excel 2007 появились *темы документов*. Полностью изменить внешний вид документа, применив к нему тему, можно единственным щелчком мыши.

Листы диаграмм

Лист диаграммы обычно содержит одну диаграмму. Эти листы игнорируются многими пользователями, которые предпочитают сохранять диаграммы на графическом слое рабочего листа. Использовать листы диаграмм необязательно, но они облегчают печать, если на странице печатается только диаграмма. Кроме того, листы диаграмм можно эффективно использовать при создании презентаций. На рис. 2.1 показан лист диаграммы, на котором находится круговая диаграмма.

Листы макросов XLM

Лист макросов XLM (который еще называется *листом макросов MS Excel 4*), в сущности, является тем же рабочим листом, но со своими стандартными настройками. В частности, на листе макросов XLM отображаются сами формулы, а не результаты их вычисления. Кроме того, стандартная ширина его столбцов больше ширины обычного рабочего листа.

Как можно понять из названия, лист макросов XLM предназначен для хранения макросов XLM. Система макросов XLM является "пережитком", доставшимся нам от предыдущих версий Excel (4.0 и более ранних). Впрочем, разработчики Excel 2010 из соображений совместимости предусмотрели поддержку макросов XLM, однако сохранить их не удастся. В этой книге система макросов XLM не рассматривается. Основное внимание уделено более мощной системе макросов VBA.

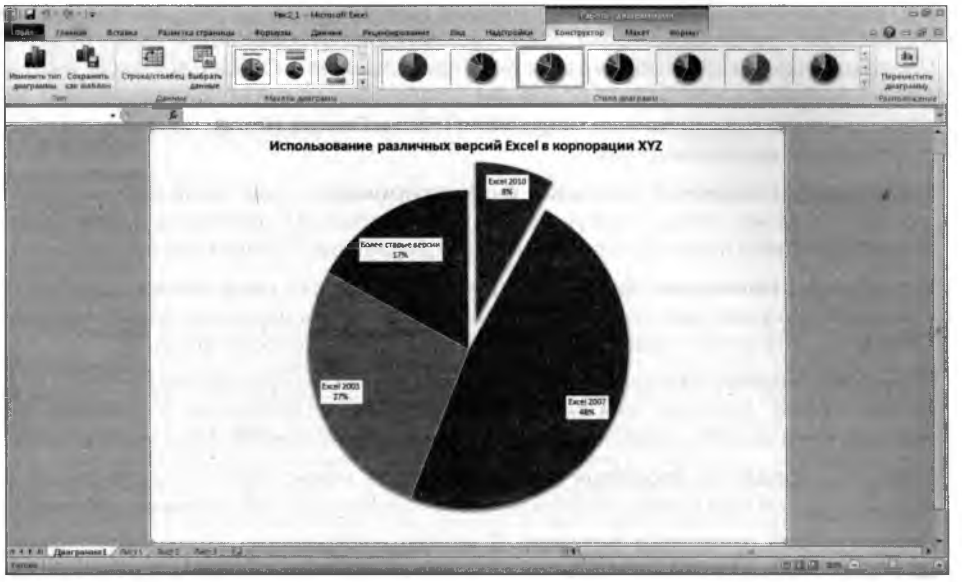


Рис. 2.1. На листе диаграммы находится круговая диаграмма

Диалоговые листы Excel 5/95

В Excel 5 и Excel 95 пользовательское диалоговое окно создавалось путем вставки специального диалогового листа. Несмотря на то что Excel 97 и более поздние версии также поддерживают применение этих листов, существует более удачная альтернатива — пользовательские формы (UserForm). В редакторе Visual Basic управление осуществляется именно пользовательскими формами.

Когда вы открываете рабочую книгу с диалоговым листом, созданным в Excel 5/95, то этот лист выглядит, как лист рабочей книги.

Учтите, что в данной книге вы не найдете информации о диалоговых листах Excel 5/95.

Что нового в Excel 2010

Ниже приводится краткий обзор новых возможностей Excel 2010.

- **64-разрядная версия**. Если ваш компьютер и версия Windows являются 64-разрядными, установите 64-разрядную версию Excel, которая позволяет создавать огромные рабочие книги. В этом случае могут возникать проблемы из-за несовместимости некоторых макросов и надстроек. Например, макросы, которые реализуют вызовы 32-разрядной библиотеки Windows API, могут не работать в 64-разрядной среде Excel 2010. Но в большинстве случаев можно изменить код таким образом, чтобы вызовы API-функций могли корректно выполняться в обеих версиях Excel.
- **Спарклайны**. Внедряемые в отдельные ячейки графики, обеспечивающие графическое представление выделенных диапазонов данных.
- **Срезы**. С помощью срезов обеспечивается новый способ фильтрации и отображения данных в сводных таблицах.
- **Новые возможности форматирования сводных таблиц**. Пользователи Excel 2010 получили большую степень контроля над внешним видом отчетов сводных таблиц.

- **Мутация кнопки Office**. Вместо большой круглой кнопки Office, использующейся в Excel 2007, появилась кнопка Файл (File), которая отображается слева от вкладок. После щелчка на ней отображается окно представления Office Backstage, в котором можно выполнять различные операции с рабочими книгами. Представление Backstage сменило традиционные меню Файл (File) и Печать (Print), но при этом обладает большими возможностями.
- **Усовершенствованное условное форматирование**. Цветовые шкалы условного форматирования теперь могут заливаться сплошным цветом, а само условное форматирование стало более точным и поддерживает отрицательные значения.
- **Усовершенствованные функции**. Улучшена точность ряда финансовых и статистических функций рабочего листа Excel. Эти функции получили новые имена, хотя в целях обеспечения совместимости остались старые версии функций.
- **Новые возможности коррекции изображений**. У пользователей Excel 2010 появились новые функции коррекции изображений, вставленных в рабочую книгу, включая возможность удаления второстепенных фрагментов фона изображения.
- **Средство создания экранных снимков**. Теперь можно легко создать снимок окна любой открытой программы, а затем вставить полученное изображение в рабочий лист.
- **Оперативный просмотр вставки**. В случае копирования диапазона данных команда Вставить (Paste) отображает различные варианты вставки в режиме оперативного просмотра. Благодаря этому вы сможете увидеть результат вставки еще до ее фактического выполнения.
- **Настройка ленты**. Теперь пользователи могут настраивать ленту путем добавления на нее новых вкладок и групп. К сожалению, настраивать ленту средствами VBA невозможно.
- **Редактор формул**. Можно создавать и вычислять математические формулы и внедрять их на рабочий лист.
- **Более быстрая работа**. Компания Microsoft внесла улучшения в механизм вычислений; также была ускорена загрузка файлов.
- **Новые возможности обеспечения безопасности**. Рабочие книги, загруженные из Интернета или полученные в виде вложений в электронные письма, открываются в защищенном режиме. Если придать рабочим книгам статус доверенных, отпадет необходимость их хранения в специальных "карантинных" папках.
- **Надстройка Поиск решения (Solver)**. В Excel 2010 появилась новая версия надстройки Поиск решения.
- **Усовершенствования в VBA**. Теперь появилась возможность с помощью макросов VBA выполнять операции, которые раньше были подвластны лишь старым макросам XLM. Помимо этого, стала доступной запись макросов, которые выполняют такие операции, как форматирование диаграмм и фигур.

Пользовательский интерфейс Excel

Пользовательский интерфейс — это средство взаимодействия между пользователем и компьютерной программой. В состав пользовательского интерфейса входят такие элементы, как меню, панели инструментов, диалоговые окна, комбинации клавиш и т.п.

С появлением версии Excel 2007 традиционные меню и панели инструментов ушли в прошлое. Современный пользовательский интерфейс Excel включает следующие элементы:

- лента;
- контекстные меню;
- диалоговые окна;
- клавиатурные сокращения;
- смарт-теги;
- области задач.

Лента

В Office 2007 компания Microsoft представила миру совершенно новый пользовательский интерфейс. Меню и панели инструментов остались в прошлом, уступив место интерфейсу пользователя, основанному на *ленте с вкладками*. Просто щелкните на вкладке, находящейся в верхней части меню (обозначена словом **Главная** (Home), **Вставка** (Insert) или **Разметка страницы** (Page Layout)), и лента отобразит команды для этой вкладки. Программа Office 2007 стала первой в истории, использующей этот новый интерфейс. В настоящее время несколько компаний внедрили этот новый стиль пользовательского интерфейса в свои продукты.

Количество команд, отображаемых на ленте, изменяется в зависимости от ширины окна Excel. Если окно слишком узкое для отображения всего, что требуется, команды адаптируются, причем создается впечатление, будто часть из них теряется. Но на самом деле это не так. На рис. 2.2 показана вкладка ленты **Главная** (Home), на которой находятся все элементы управления. На рис. 2.3 показана лента для более узкого окна Excel. Обратите внимание на отсутствие части описательного текста при сохранении всех пиктограмм. На рис. 2.4 показан экстремальный случай, когда окно чрезвычайно узкое. Некоторые из групп содержат единственную пиктограмму, после щелчка на которой отображаются все команды для этой группы.

Рис. 2.2. Вкладка ленты Главная

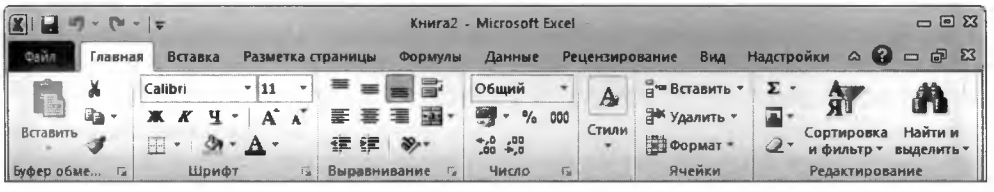


Рис. 2.3. Вкладка ленты Главная для суженного окна Excel

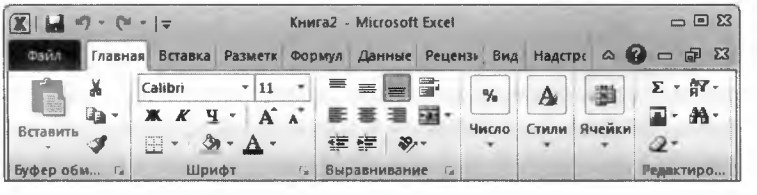


Рис. 2.4. Вкладка ленты Главная для самого узкого окна Excel

Совет

Если нужно скрыть ленту для увеличения рабочего пространства, дважды щелкните на любой из вкладок. После этого лента исчезнет, а в вашем распоряжении окажутся примерно четыре дополнительные строки рабочего листа. Если вы захотите воспользоваться лентой снова, просто щелкните на любой из вкладок. Для отображения/сокрытия ленты можно также нажать комбинацию клавиш <Ctrl+F1> либо щелкнуть на пиктограмме в виде направленной вверх стрелки, находящейся слева от пиктограммы справки в верхней правой части окна.

Контекстные вкладки

Помимо обычных вкладок в Excel имеются *контекстные вкладки*. Подобная вкладка содержит набор инструментов, отображаемый при выделении какого-либо объекта (диаграмма, таблица, рисунок либо графический элемент SmartArt).

На рис. 2.5 показаны контекстные вкладки, которые появляются в случае выделения встроенной формулы. При этом Excel отображает две контекстные вкладки: **Формат** (Format) (для работы с объектами) и **Конструктор** (Design) (для работы с уравнениями). Обратите внимание на то, что в строке заголовка Excel отображается описание групп контекстных вкладок (**Средства рисования** (Drawing Tools) и **Работа с формулами** (Equation Tools)). В случае отображения контекстных вкладок можно продолжать пользоваться всеми остальными вкладками.

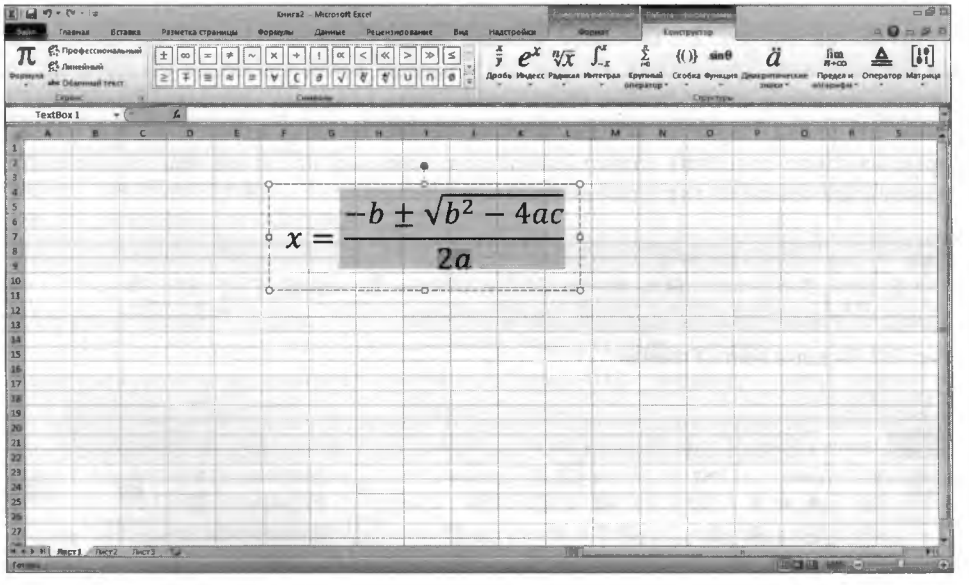


Рис. 2.5. После выделения объекта отображаются контекстные вкладки, содержащие инструменты для работы с этим объектом

Типы команд ленты

В большинстве случаев команды ленты выполняют то, чего вы от них ожидаете. Ниже описаны основные команды, на практике же встречаются различные стили этих команд.

- **Простые кнопки.** Щелкните на кнопке для выполнения связанной с ней команды. Примером может служить кнопка **Увеличить размер шрифта** (Increase Font

Size) в группе Шрифт (Font) вкладки Главная (Home). Одни кнопки выполняют связанное с ними действие немедленно; другие отображают диалоговое окно, в котором можно ввести дополнительную информацию. Кнопки могут также сопровождаться текстом.

- **Переключатели.** Переключатель допускает щелчок мышью, а также отображает некоторый тип информации двумя цветами. Примером подобной кнопки является кнопка Полужирный (Bold) в группе Шрифт (Font) вкладки Главная (Home). Если активная ячейка не выделена полужирным, кнопка Полужирный окрашена в обычный цвет. Если же активная ячейка выделена полужирным, эта кнопка окрашена в другой фоновый цвет. После щелчка на этой кнопке переключается атрибут Полужирный для выделенной области.
- **Простые списки.** Если команде ленты соответствует маленькая указующая вниз стрелка, значит, тип этой команды — список. Щелкните на нем, и ниже появятся дополнительные команды. Примером простого списка может служить команда Объединить и поместить в центре (Merge and Center) в группе Выравнивание (Alignment) вкладки Главная (Home). После щелчка на этом элементе управления отобразятся четыре элемента, обеспечивающие центрирование и объединение данных.
- **Комбинированные кнопки.** Этот вид элемента управления представляет собой комбинацию обычной кнопки (в верхней части) и раскрывающегося списка (в нижней части). После щелчка в верхней части вызывается соответствующая команда. Если щелкнуть на нижней части, отображается список связанных команд. Идентифицировать комбинированную кнопку можно по двум цветам, отображающимся при установке указателя мыши над этой кнопкой. В качестве примера можно рассмотреть команду Вставить (Paste), находящуюся в группе Буфер обмена (Clipboard) ленты Главная (Home). После щелчка на верхней части этой кнопки вставляются данные, находящиеся в буфере обмена. Если же щелкнуть на нижней части, отображается перечень команд, связанных с операцией вставки (рис. 2.6).
- **Флажки.** Флажок позволяет выбрать состояние "включено/выключено". Примером может служить флажок Сетка (Gridlines) в группе Показать или скрыть (Show/Hide) вкладки Вид (View). Если этот флажок установлен, на листе отображаются линии сетки. Если этот флажок не установлен, сетка не отображается.
- **Счетчики.** Примеры счетчиков можно найти в группе Вписать (Scale to Fit) вкладки Разметка страницы (Page Layout). Щелкните на верхней части счетчика для увеличения его значения; щелкните на нижней части счетчика для уменьшения его значения.

Перекрестная ссылка

Дополнительные сведения о настройке ленты Excel можно найти в главе 22.

В правой части некоторых групп ленты находятся маленькие значки, с помощью которых открываются диалоговые окна. Например, подобный значок находится в группе Выравнивание (Alignment) вкладки Главная (Home) (рис. 2.7). После щелчка на этом значке отображается диалоговое окно Формат ячеек (Format Cells) с выбранной вкладкой Число (Number). Здесь можно найти параметры, отсутствующие на ленте.

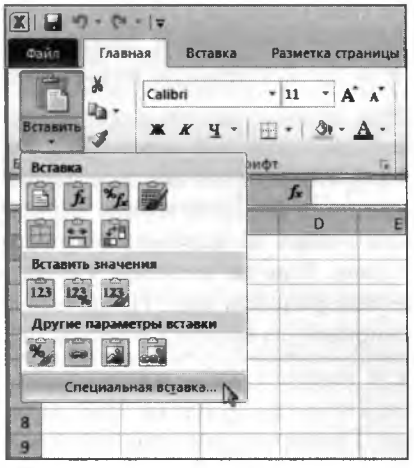


Рис. 2.6. Команде Вставить соответствует комбинированная кнопка

Новинка

Пользователи Excel 2007 не могли изменять ленту по своему усмотрению, тогда как пользователи Excel 2010 получили возможность добавлять и удалять команды ленты. Дополнительные сведения о настройке ленты можно найти в главе 22.

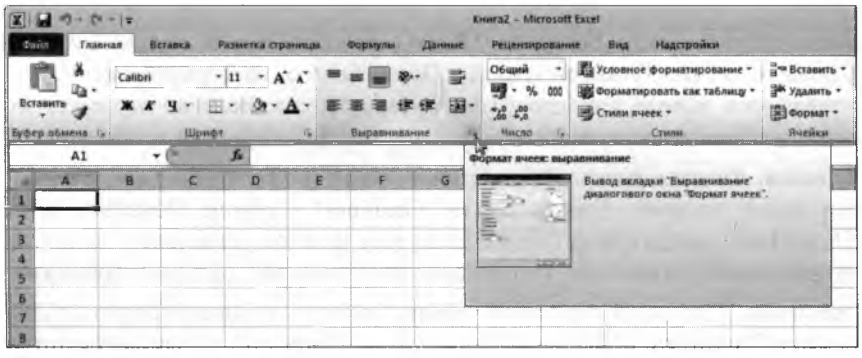


Рис. 2.7. После щелчка на этом маленьком значке отображается окно, содержащее дополнительные параметры

Панель быстрого доступа

На панели быстрого доступа находятся чаще всего используемые команды. Эта панель всегда отображается на экране независимо от выбранной в данный момент вкладки ленты. Обычно панель быстрого доступа находится в левой части строки заголовка. При желании ее можно переместить ниже ленты. Для этого щелкните на ней правой кнопкой мыши и в контекстном меню выберите команду **Разместить панель быстрого доступа под лентой** (Show Quick Access Toolbar Below the Ribbon).

Изначально на панели быстрого доступа находятся следующие команды: **Сохранить** (Save), **Отменить** (Undo) и **Повторить** (Redo). Можно также добавить другие команды, которые наиболее часто используются в повседневной практике. Если нужно перемес-

тить команду с ленты на панель быстрого доступа, щелкните на этой команде правой кнопкой мыши и выберите пункт меню **Добавить на панель быстрого доступа** (Add To Quick Access Toolbar).

В Excel существуют команды, которые не отображаются на ленте. Единственный способ воспользоваться этими командами — добавить их на панель быстрого доступа. Обратите внимание на рис. 2.8, на котором показан раздел **Панель быстрого доступа** (Quick Access toolbar) диалогового окна **Параметры Excel** (Excel Options). С помощью этого раздела обеспечивается быстрая настройка панели быстрого доступа. Для быстрого перехода в раздел щелкните правой кнопкой мыши на панели быстрого доступа и выберите пункт **Настройка панели быстрого доступа** (Customize Quick Access Toolbar).

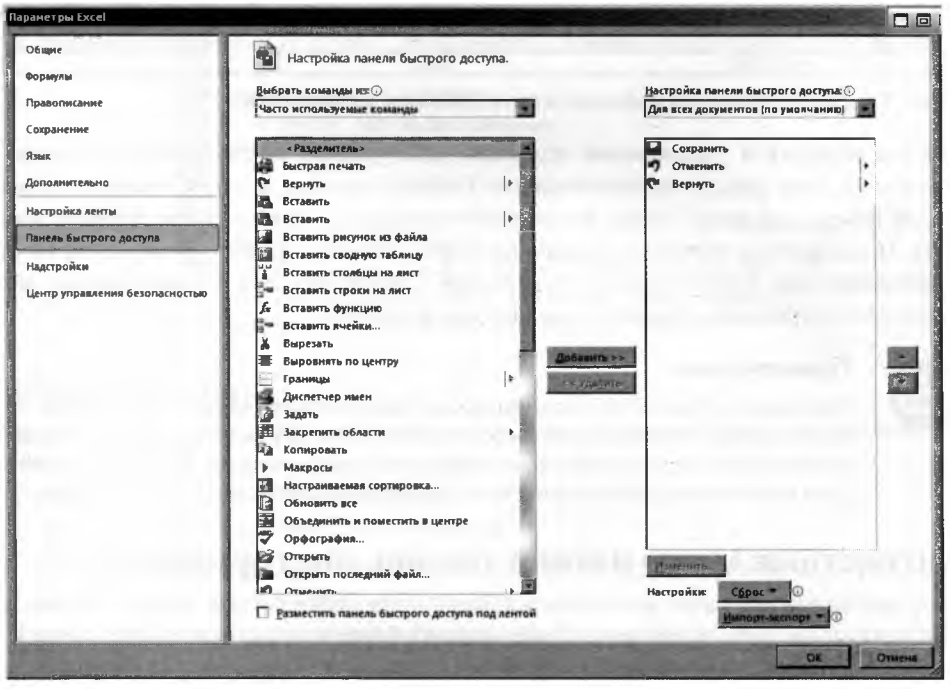


Рис. 2.8. Добавление новых значков на панель быстрого доступа с помощью раздела Панель быстрого доступа диалогового окна Параметры Excel

Быстрый доступ к ленте с помощью клавиатуры

На первый взгляд кажется, что работать с лентой можно исключительно с помощью мыши. Это впечатление подкрепляется еще и тем, что в названиях команд отсутствуют подчеркнутые буквы, соответствующие клавишам, с помощью которых (в комбинации с клавишей <Alt>) вызываются эти команды. На самом деле лента тесно "дружит" с клавиатурой. Для отображения подсказок клавиш нажмите клавишу <Alt>. Каждому элементу управления, находящемуся на ленте, соответствует своя клавиша.

Совет

В процессе ввода букв, соответствующих подсказке клавиши, вовсе не обязательно нажимать клавишу <Alt>.

На рис. 2.9 показана вкладка **Главная** (Home) после нажатия клавиши <Alt> (для отображения подсказок клавиш). Если нажать клавишу, отображаемую одной из подсказок клавиш, отобразятся дополнительные подсказки клавиш. Например, если с помощью клавиатуры нужно выровнять по левому краю содержимое ячейки, нажмите сначала клавишу <Alt>, затем — клавишу <H> (для выбора вкладки **Главная** (Home)), после чего — сочетние клавиш <A> и <L> (для выравнивания влево (Align Left)). Если вы относитесь к категории компьютерных фанатов (вроде меня), то через несколько минут вспомните комбинацию клавиш для вызова той или иной команды.

Рис. 2.9. Для отображения подсказок клавиш нажмите клавишу <Alt>

После нажатия и удерживания клавиши <Alt> можно воспользоваться клавишами <←> и <→> для прокручивания вкладок. Выбрав нужную вкладку, нажмите клавишу <↓> для перехода к ленте. Также используйте клавиши <←> и <→> для перебора команд ленты. После выбора нужной команды нажмите клавишу <Enter>. Этот метод не столь эффективный, как использование комбинаций клавиш, но зато с его помощью можно быстро просмотреть все варианты, предлагаемые лентой.

Примечание

Программа Excel 2010 поддерживает комбинации клавиш Excel 2003, предназначенных для быстрого вызова меню. Поэтому вы, как и раньше, сможете воспользоваться подобными комбинациями клавиш, например <Alt+ES> (для перехода в диалоговое окно Специальная вставка (Paste Special)).

Контекстные меню и мини-панель инструментов

Единственный вид меню, оставшийся в Excel, — это контекстные меню. Этот тип меню вызывается после выбора одного или нескольких объектов и щелчка правой кнопкой мыши. Эти меню контекстно-зависимы. Другими словами, отображаемое содержимое на экране меню зависит от того, в каком месте экрана находится указатель мыши в данный момент. Правой кнопкой мыши можно щелкнуть практически везде — на ячейке, на границе строки или столбца, на строке заголовка рабочей книги, на панели инструментов и т.п.

После щелчка правой кнопкой мыши на некоторых объектах над контекстным меню отображается мини-панель инструментов. С ее помощью обеспечивается доступ к наиболее распространенным командам форматирования. На рис. 2.10 показана мини-панель инструментов, появляющаяся при выборе ячейки.

Хотя с помощью VBA невозможно настроить ленту, средствами этого языка можно настроить любое из контекстных меню. Также с помощью VBA невозможно изменить внешний вид мини-панели инструментов.

Перекрестная ссылка

Дополнительные сведения о настройке контекстных меню можно найти в главе 23.

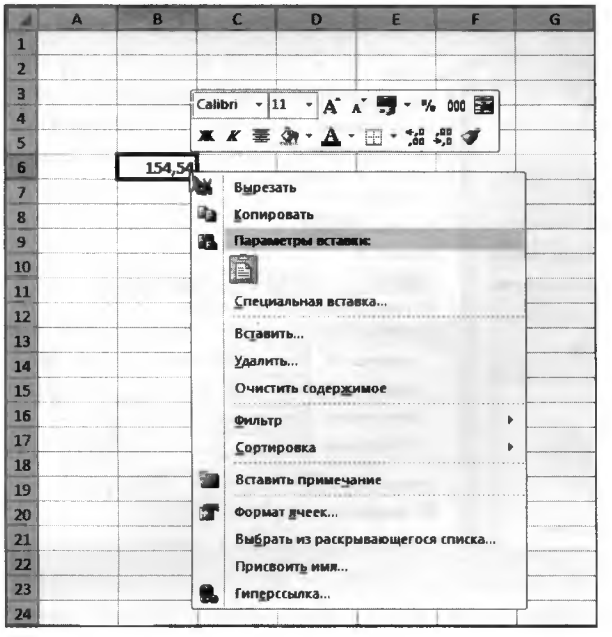


Рис. 2.10. После щелчка правой кнопкой мыши на некоторых объектах отображается мини-панель инструментов

Диалоговые окна

При вызове на выполнение некоторых команд ленты отображаются диалоговые окна. Во многих случаях эти окна включают дополнительные элементы управления, которые недоступны при использовании одной ленты.

В Excel существуют два вида диалоговых окон.

- **Модальные диалоговые окна.** Если на экране отображается модальное диалоговое окно, при вызове какой-либо команды его нужно закрыть. Примером подобного окна может служить диалоговое окно **Формат ячеек** (Format Cells). Если в нем выбран какой-либо параметр, его невозможно применить до тех пор, пока вы не щелкнете на кнопке **OK**. Если изменения не предусмотрены, щелкните на кнопке **Отмена** (Cancel) для закрытия диалогового окна.
- **Немодальные диалоговые окна.** Эти диалоговые окна всегда находятся поверх других окон. Например, если вы работаете с диаграммой и используете диалоговое окно **Формат** (Format), выполняемые в этом окне настройки немедленно отразятся на диаграмме. Эти диалоговые окна практически всегда содержат кнопку **Закрыть** (Close) вместо кнопки **OK**, а также кнопку **Отмена** (Cancel).

Многие из диалоговых окон Excel напоминают записную книжку, в результате чего одно диалоговое окно заменяет несколько окон. В прежних диалоговых окнах эти вкладки находились в верхней части. В современном диалоговом окне (пример которого показан на рис. 2.11) вкладки находятся слева.

Разработчики могут создавать пользовательские диалоговые окна с помощью элемента `UserForm` (пользовательские формы). В дальнейшем будет рассмотрено создание самых разных диалоговых окон, включая диалоговые окна с вкладками.

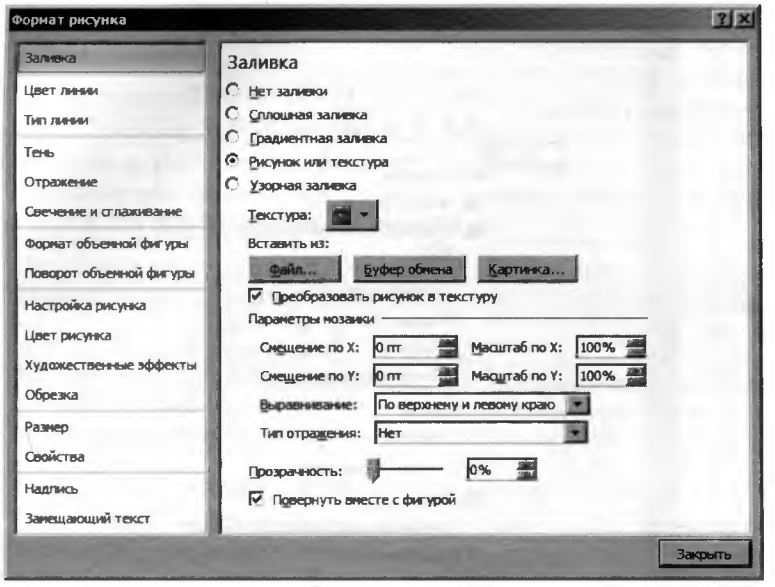


Рис. 2.11. Диалоговые окна с вкладками обеспечивают простой доступ к находящимся на них параметрам

Перекрестная ссылка

Дополнительные сведения о пользовательских формах можно найти в части IV.

Комбинации клавиш

Пользователям программы Excel доступно *множество* комбинаций клавиш. Например, если нужно скопировать содержимое ячейки в находящуюся ниже выделенную ячейку, нажмите <Ctrl+D>. Если вы начинающий пользователь Excel или хотите повысить скорость выполнения операций в программе, просмотрите разделы справочной системы, посвященные комбинациям клавиш. Знание комбинаций клавиш — ключ к успешному использованию возможностей Excel. В файлах справочной системы приведены таблицы, в которых собраны все полезные советы по использованию клавиатуры для реализации часто выполняемых операций.

И как уже отмечалось, с помощью клавиатуры можно получить доступ к командам ленты.

Смарт-теги

Смарт-тег — это небольшой значок, который автоматически появляется на рабочем листе при выполнении определенных действий. После щелчка на смарт-теге на экране отображается своеобразное контекстное меню, в котором перечислены специальные команды. Например, при копировании и вставке диапазона ячеек на рабочий лист вы увидите смарт-тег в правом нижнем углу вставляемого диапазона (рис. 2.12).

Область задач

Впервые *область задач* появилась в Excel 2002. Это многофункциональное средство, прикрепляемое к правому краю окна, значительно улучшило пользовательский интерфейс программы. Область задач используется для выполнения самых разных операций.

Среди них — отображение содержимого буфера обмена, вывод списка полей сводной таблицы, вставка клипов, упрощение поиска, а также установка соответствия с данными XML (eXtensible Markup Language). Посмотрите на рис. 2.13, на котором показана область задач Картинка (Clip Art).

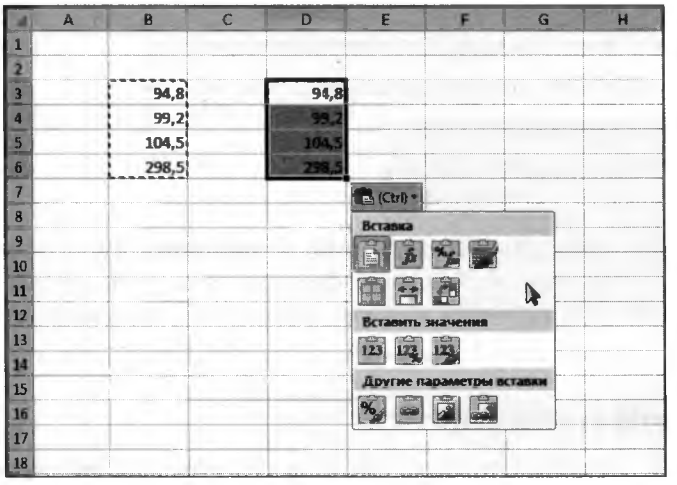


Рис. 2.12. Смарт-тег, отображаемый в процессе вставки скопированного диапазона ячеек

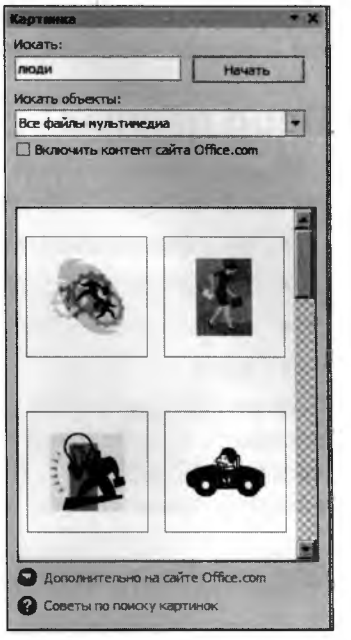


Рис. 2.13. Поиск клипов — это лишь одна из многочисленных функций области задач Картинка

Что нового в Visual Basic Editor

А ничего.

Доступ к большинству компонентов обновленной объектной модели Excel 2010 обеспечивается посредством кода VBA, хотя модуль VB Editor остался точно таким же, как и в предыдущих версиях Excel. Все входящие в состав Microsoft Office приложения используют новый ленточный интерфейс пользователя, а работа с редактором VB Editor по-прежнему осуществляется с помощью устаревших меню и панелей инструментов. Возможно, в будущих версиях VB Editor мы увидим обновленное меню, но пока что все остается без изменений.

Настройка окна программы

Окно Excel на ходу перестраивается в зависимости от типа отображаемых данных (строка состояния, строка формул, панели инструментов и т.д.). Соответствующие команды находятся на вкладке Вид (View).

Фактически с помощью Excel можно отображать данные таким образом, что они будут совершенно не похожи на электронную таблицу. Например, если выбрать команду Вид⇨Режимы просмотра книги⇨Полноэкранный режим (View⇨Workbook Views⇨ Full Screen), на экране останется только строка заголовка. Этот режим просмотра следует

выбирать в том случае, когда нужно вывести на экран как можно больше информации. Для выхода из этого режима щелкните правой кнопкой мыши на любой ячейке и в контекстном меню выберите команду Вернуть обычный режим (Close Full Screen).

Обратите внимание на ползунок изменения масштаба, отображаемый в правой части строки состояния. С помощью этого ползунка можно легко масштабировать изображение на экране. Помимо этого, можно щелкнуть правой кнопкой мыши на строке состояния и выбрать тип просматриваемой информации.

Ввод данных

Вводить данные в среде Excel достаточно просто. Каждое введенное в ячейку значение интерпретируется программой Excel как один из следующих элементов:

- числовое значение (им может быть значение даты и/или времени);
- текст;
- булево значение ("истина" или "ложь");
- формула.

Формулы начинаются знаком равенства (=). При этом разработчики Excel учли склонность пользователей Lotus 1-2-3 вводить в начале формулы знак "собачки" (@), плюс (+) либо минус (–). Эти знаки автоматически преобразуются в знак равенства после нажатия клавиши <Enter>.

Формулы, функции и имена

Именно благодаря формулам создается структура, именуемая электронной таблицей. Программа Excel включает ряд связанных с формулами свойств, которые не слишком известны широкой публике. Эти свойства обеспечивают создание массивов формул, включение оператора пересечения, ссылок, а также создание *мегаформул* (авторский термин, означающий длинные и малопонятные, но очень эффективные формулы).

Перекрестная ссылка

Подробнейшее описание формул ожидает вас в главе 3.

Приложение Excel включает ряд полезных свойств, облегчающих идентификацию ошибок, а также отслеживание логики в громоздких электронных таблицах. Для получения доступа к этим свойствам воспользуйтесь командами в группе Формулы⇨Зависимости формул (Formulas⇨Formula Auditing).

Весьма полезной является команда Формулы⇨Зависимости формул⇨Проверка наличия ошибок (Formulas⇨Formula Auditing⇨Error Checking). В процессе ее выполнения осуществляется сканирование рабочего листа с целью выявления "неправильных" формул. На рис. 2.14 показано, что Excel идентифицирует некорректную формулу, а также предлагает пользователю ряд решений, позволяющих устранить обнаруженную проблему.

Благодаря использованию функций рабочего листа обеспечивается выполнение вычислений и операций, которые были бы невозможны при использовании других методов. Программа Excel поддерживает огромное количество встроенных функций.

Простейший способ найти требуемую функцию заключается в обращении к диалоговому окну Мастер функций (Insert Function), которое показано на рис. 2.15. Для получе-

ния доступа к этому окну щелкните на кнопке **Вставить функцию** (Insert Function), которая находится в строке формул, либо нажмите комбинацию клавиш <Shift+F3>. После этого на экране появится диалоговое окно **Аргументы функции** (Function Arguments), которое облегчает назначение функциональных аргументов.

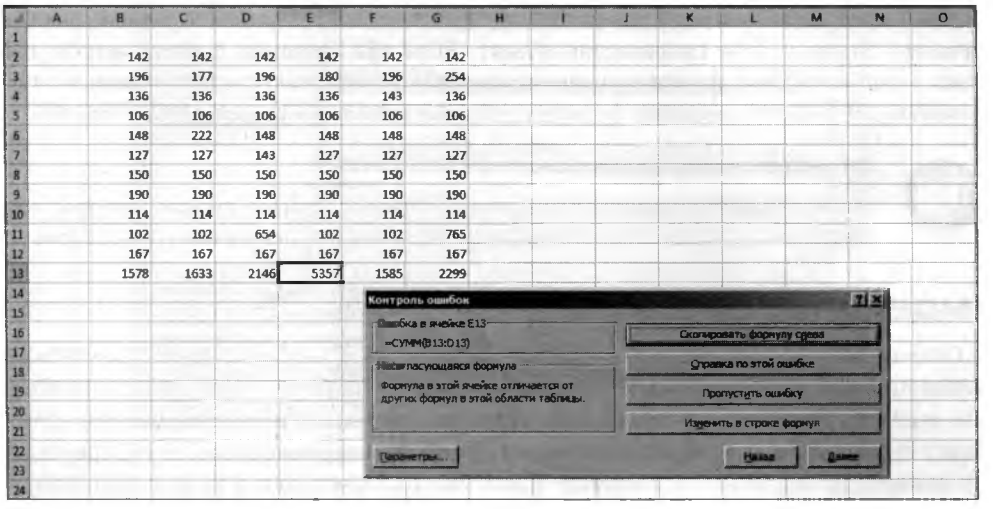


Рис. 2.14. Программа Excel может выявлять ошибки в формулах

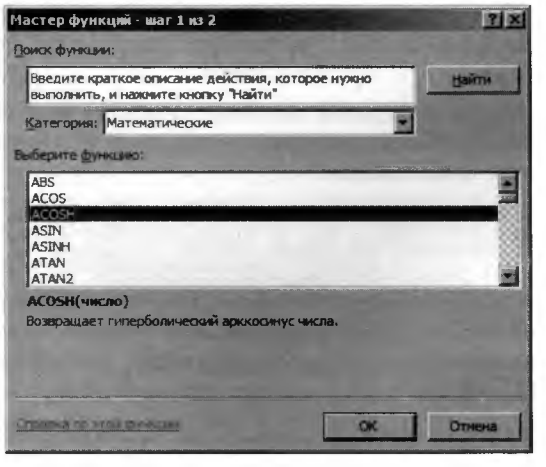


Рис. 2.15. Наилучший способ включения функции в формулу — воспользоваться этим окном

Примечание

Начиная с версии Excel 2007 функции, ранее входившие в состав надстройки Analysis ToolPak (Пакет анализа), получили статус "встроенных". Другими словами, вы можете получить доступ к этим функциям, даже если надстройка Analysis ToolPak не установлена.

Перекрестная ссылка

Пользователи Excel могут также создавать собственные функции рабочего листа с помощью средств VBA. Дополнительные сведения по этой теме приведены в главе 10.

Имя — это идентификатор, с помощью которого можно ссылаться на ячейку, диапазон, значение, формулу либо графический объект. Если при создании формул используются имена, подобные формулы легче воспринимаются (по сравнению с использованием ссылок на ячейки). Еще проще создавать формулы, которые используют именованные ссылки.

Перекрестная ссылка

Имена подробно рассматриваются в главе 3. Там же описан процесс обработки имен, который происходит несколькими уникальными способами.

Выделение объектов

Обычно объекты выделяются с помощью стандартных методов, принятых в Windows. Диапазон ячеек можно выделить с помощью мыши, щелкнув и затем обведя необходимые ячейки. Если щелкнуть на объекте, который расположен на графическом слое, то объект будет выделен. Чтобы выделить ряд объектов или несмежных ячеек, при выделении каждого из них нажмите клавишу <Ctrl>. Если следует выделить большой диапазон, щелкните на ячейке, расположенной в одном из углов этого диапазона, прокрутите документ до противоположного угла диапазона, а затем, нажав клавишу <Shift>, щелкните мышью на последней ячейке диапазона.

Примечание

После щелчка на диаграмме выделяется объект в составе диаграммы. Если же нужно выделить всю диаграмму, удерживайте нажатой клавишу <Ctrl> во время щелчка на диаграмме.

Если объекту сопоставлен макрос, этот макрос будет запущен после щелчка на объекте. Если нужно просто выделить подобный объект, щелкните на нем правой кнопкой мыши и нажмите клавишу <Esc> для сокрытия контекстного меню либо во время щелчка на объекте нажимайте клавишу <Ctrl>.

Форматирование

В Excel выполняется форматирование двух видов: числовое и стилистическое.

Числовое форматирование определяет вид, который принимает значение в ячейке. Вы можете не только выбрать формат из заранее определенного списка, но и создать собственный формат (рис. 2.16). Эта процедура подробно описана в справочной системе программы.

Некоторые числовые форматы задаются автоматически в зависимости от вводимого значения. Например, если введено значение с принятым у вас символом валюты (в США таким символом является знак доллара), то будет использован числовой денежный формат. Можно также воспользоваться возможностью условного числового форматирования, которое определяет вид ячейки с числовым значением в зависимости от величины этого значения.

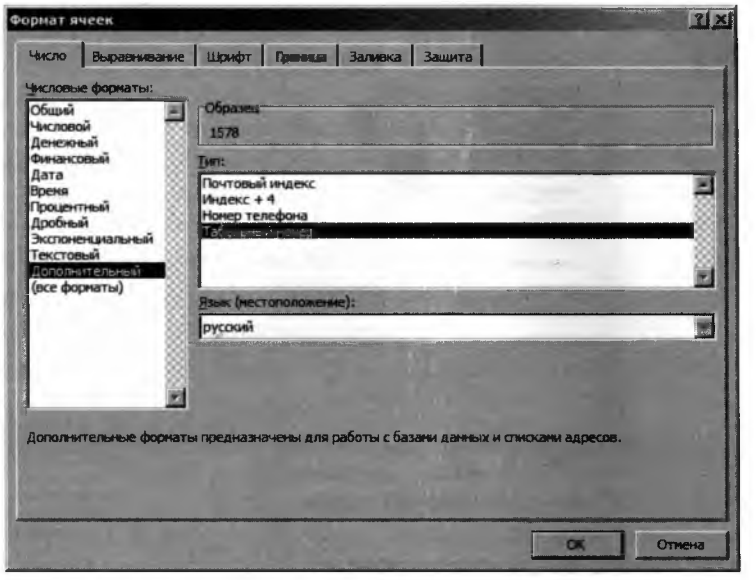


Рис. 2.16. Параметры числового форматирования в Excel весьма гибкие

Стилистическим называется форматирование, применяемое с целью улучшения внешнего вида листа. Многие кнопки на вкладках ленты обеспечивают прямой доступ к основным возможностям форматирования, но иногда требуется доступ к диалоговому окну **Формат** (Format) для выбранного объекта, в котором содержится весь спектр параметров форматирования.

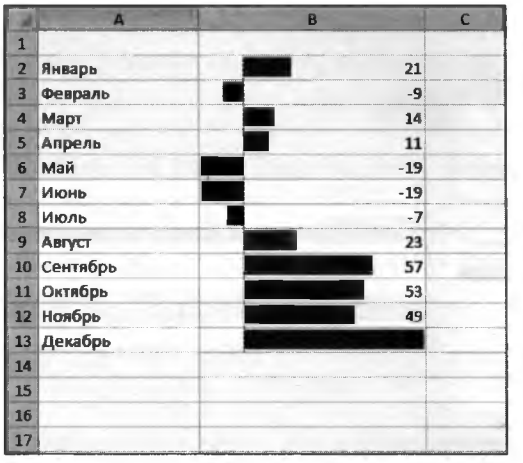
Простейший способ вывести необходимое диалоговое окно и задать формат объекта — выделить объект и нажать комбинацию клавиш <Ctrl+1>. Можете также щелкнуть на объекте правой кнопкой мыши и выбрать из контекстного меню команду `Формат` *xxx* (Format *xxx*), где *xxx* — это название выделенного объекта. В результате появится диалоговое окно с несколькими вкладками. В нем можно задать любое форматирование, какое только можно присвоить выделенному объекту.

В Excel часто используется такая возможность, как условное форматирование. Доступ к соответствующим параметрам можно получить, выполнив команду **Главная⇨Стили⇨Условное форматирование** (Home⇨Styles⇨Conditional Formatting). Эта средство позволяет задать форматирование, которое будет применяться только при определенных условиях. Например, можно выделить другим цветом те ячейки, значения в которых превышают указанную величину.

В Excel 2007 появилось много новых возможностей условного форматирования, включая гистограммы, цветовые шкалы и наборы значков. Первая из этих возможностей иллюстрируется на рис. 2.17 — непосредственно в ячейках отображается гистограмма условного формата.

Параметры защиты

В Excel предлагается несколько способов защиты данных рабочего листа. Например, можно защитить от изменения либо перезаписи только формулы, структуру рабочей книги или VBA-код.



	A	B	C
1			
2	Январь	21	
3	Февраль	-9	
4	Март	14	
5	Апрель	11	
6	Май	-19	
7	Июнь	-19	
8	Июль	-7	
9	Август	23	
10	Сентябрь	57	
11	Октябрь	53	
12	Ноябрь	49	
13	Декабрь		
14			
15			
16			
17			

Рис. 2.17. Условное форматирование, выполняемое с помощью гистограмм

Защита формул от перезаписи

Во многих случаях нужно защитить формулы от перезаписи или изменения. Для этого выполните следующие действия.

1. Выберите ячейки с формулами, которые нужно защитить от изменения.
2. Щелкните на них правой кнопкой мыши и в контекстном меню выберите команду **Формат ячеек** (Format Cells).
3. Находясь в диалоговом окне **Формат ячеек** (Format Cells), выберите вкладку **Защита** (Protection).
4. На вкладке **Защита** отмените установку флажка **Защищаемая ячейка** (Locked).
5. Щелкните на кнопке **OK** для закрытия диалогового окна **Формат ячеек**.
6. Выполните команды **Рецензирование⇨Изменения⇨Защитить лист** (Review⇨Changes⇨Protect Sheet). После этого появится диалоговое окно **Защита листа** (Protect Sheet), показанное на рис. 2.18.
7. В диалоговом окне **Защита листа** укажите параметры, которые обеспечивают выполнение соответствующих действий, введите пароль (при необходимости) и щелкните на кнопке **OK**.

Примечание

По умолчанию все ячейки заблокированы (защищены). Подобная блокировка не дает эффекта до тех пор, пока защищен рабочий лист.

Можно также скрыть формулы, обычно отображаемые в строке формул Excel при выделении ячейки. Для этого выделите ячейки с формулами и убедитесь в том, что установлен флажок **Скрыть формулы** (Hidden) на вкладке **Защита** (Protection) диалогового окна **Формат ячеек** (Format Cells).

Защита структуры рабочей книги

Если структура рабочей книги защищена, вы не сможете добавлять либо удалять рабочие листы. Для начала следует открыть диалоговое окно **Защита структуры и окон** (Protect Structure and Windows), воспользовавшись командой **Рецензирование⇨Изменения⇨Защитить книгу** (Review⇨Changes⇨Protect Workbook). Результат наших усилий можно увидеть на рис. 2.19. Установите флажок **Структуру** (Structure). Если при этом также будет установлен флажок **Окна** (Windows), вы не сможете перемещать либо изменять размеры окна рабочей книги.

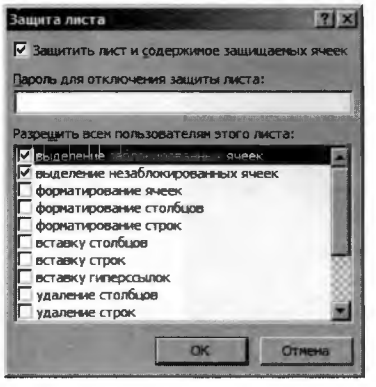


Рис. 2.18. В этом окне можно установить защиту для своего рабочего листа

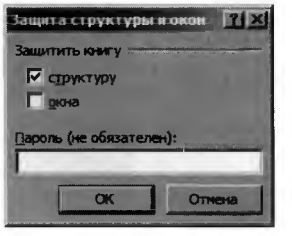


Рис. 2.19. Диалоговое окно Защита структуры и окон

Защита книги с помощью пароля

В некоторых случаях нужно блокировать доступ к книге для сторонних пользователей, воспользовавшись паролем.

Для сохранения файла рабочей книги с паролем выберите команду **Файл⇨Сведения⇨Защитить книгу⇨Зашифровать паролем** (File⇨Info⇨Protect Workbook⇨Encrypt With Password). В появившемся на экране окне **Шифрование документа** (Encrypt Document) укажите пароль и щелкните на кнопке **OK** (рис. 2.20). После этого сохраните рабочую книгу.

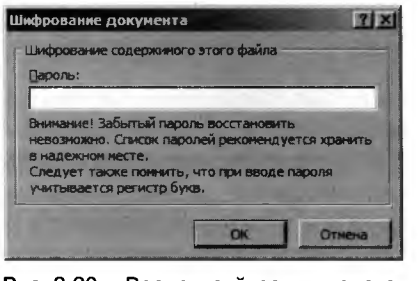


Рис. 2.20. Воспользуйтесь диалоговым окном Шифрование документа для сохранения рабочей книги с паролем

Защита VBA-кода с помощью пароля

Если в состав рабочей книги входит VBA-код, можно воспользоваться паролем для защиты этого кода от просмотра и изменения. Для этого перейдите в среду VBE (нажмите комбинацию клавиш <Alt+F11>) и выберите проект в окне Projects (Проекты). Выполните команду Tools⇨*xxx* Properties (здесь *xxx* — имя проекта). На экране появится диалоговое окно свойств проекта.

Находясь в окне свойств проекта, выберите вкладку Protection (Защита) (рис. 2.21). Установите флажок Lock Project for Viewing (Защитить проект от просмотра) и дважды введите пароль. Щелкните на кнопке OK и сохраните файл. Пароль понадобится для просмотра либо изменения кода при повторном открытии файла.

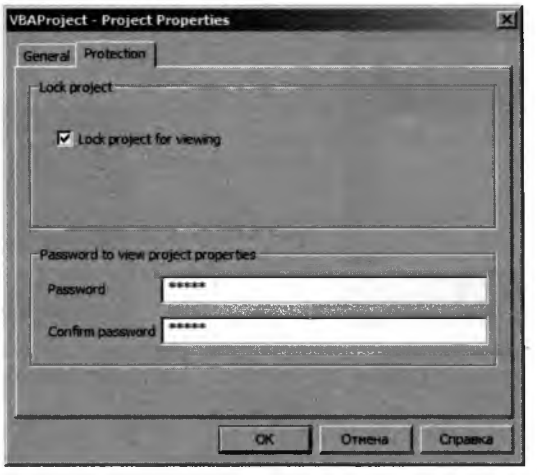


Рис. 2.21. Защита VBA-проекта в диалоговом окне Project Properties (Свойства проекта)

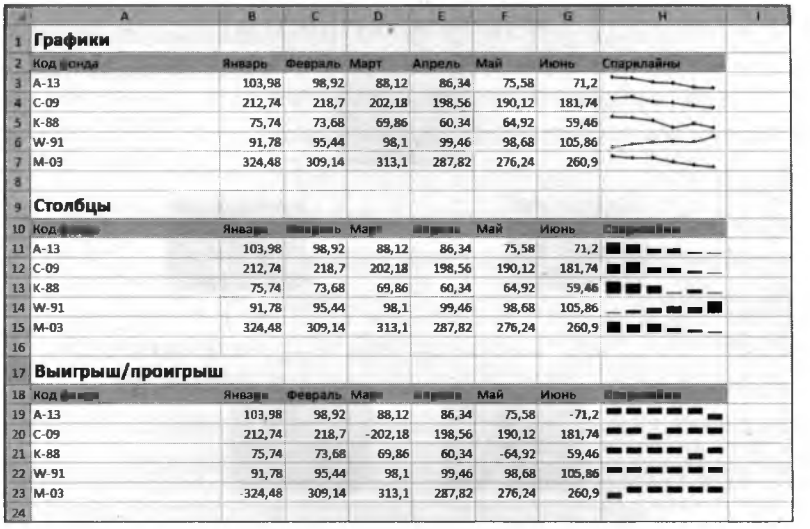
Предупреждение

Важно знать, что безопасность данных в Excel поддерживается на недостаточно высоком уровне. Возможности защиты (даже при наличии пароля), конечно, предотвратят доступ к важным данным в рабочей книге со стороны случайного пользователя, но они не спасут от посягательств опытного хакера, вооруженного программами взлома паролей (либо знающего парочку секретов программы).

Диаграммы

Программа Excel наиболее часто применяется для создания диаграмм. Как упоминалось ранее, диаграммы могут находиться на листах диаграмм либо "расплываться" на весь рабочий лист. Вы также имеете возможность создавать *диаграммы сводных таблиц*. Такая диаграмма связана с соответствующей сводной таблицей и позволяет просматривать в графическом виде сводные данные — при этом используются те же методы управления, что и в самой сводной таблице.

Пользователи Excel 2010 получили возможность создавать диаграммы нового типа — *спарклайны*. Эти небольшие диаграммы полностью помещаются в одной ячейке. Не путайте их со стандартными диаграммами Excel. На рис. 2.22 показан рабочий лист, на котором находятся спарклайны различных типов.



	A	B	C	D	E	F	G	H	I
1	**Графики**								
2	Код [illegible]онда	Январь	Февраль	Март	Апрель	Май	Июнь	Спарклайны	
3	A-13	103,98	98,92	88,12	86,34	75,58	71,2		
4	C-09	212,74	218,7	202,18	198,56	190,12	181,74		
5	K-88	75,74	73,68	69,86	60,34	64,92	59,46		
6	W-91	91,78	95,44	98,1	99,46	98,68	105,86		
7	M-03	324,48	309,14	313,1	287,82	276,24	260,9		
8									
9	**Столбцы**								
10	Код [illegible]	Янва[illegible]	[illegible]ь	Мар[illegible]	[illegible]	Май	Июнь	[illegible]	
11	A-13	103,98	98,92	88,12	86,34	75,58	71,2		
12	C-09	212,74	218,7	202,18	198,56	190,12	181,74		
13	K-88	75,74	73,68	69,86	60,34	64,92	59,46		
14	W-91	91,78	95,44	98,1	99,46	98,68	105,86		
15	M-03	324,48	309,14	313,1	287,82	276,24	260,9		
16									
17	**Выигрыш/проигрыш**								
18	Код [illegible]	Янва[illegible]	Февраль	Мар[illegible]	[illegible]	Май	Июнь	[illegible]	
19	A-13	103,98	98,92	88,12	86,34	75,58	-71,2		
20	C-09	212,74	218,7	-202,18	198,56	190,12	181,74		
21	K-88	75,74	73,68	69,86	60,34	-64,92	59,46		
22	W-91	91,78	95,44	98,1	99,46	98,68	105,86		
23	M-03	-324,48	309,14	313,1	287,82	276,24	260,9		
24									

Рис. 2.22. Рабочий лист с добавленными спарклайнами

Фигуры и рисунки SmartArt

Как уже отмечалось, каждый рабочий лист включает невидимый графический слой, на котором находятся диаграммы, рисунки, элементы управления (например, кнопки и списки), а также фигуры.

Программа Excel позволяет легко нарисовать огромное количество геометрических фигур непосредственно на рабочем листе. Для получения доступа к коллекции фигур воспользуйтесь командой Вставка⇨Иллюстрации⇨Фигуры (Insert⇨Illustrations⇨Shapes). Эти фигуры могут изменяться произвольным образом и допускают добавление текста. Можно также сгруппировать объекты, образовав единый объект, который проще перемещать, а также изменять его размеры.

В пакете Office 2007 появился новый вид иллюстраций — рисунки SmartArt. Благодаря этим рисункам обеспечивается создание настраиваемых диаграмм. Пример рисунка SmartArt показан на рис. 2.23.

Доступ к базам данных

В течение многих лет процессоры электронных таблиц предоставляли пользователям возможность работать с простыми таблицами реляционных баз данных. Не является исключением и программа Excel — она также предлагает инструменты управления реляционными базами данных.

Базы данных, создаваемые в процессорах электронных таблиц, делятся на две категории.

- **Базы данных рабочих листов.** Вся база данных хранится на рабочем листе, ограничивающем ее размеры.
- **Внешние базы данных.** Данные хранятся в одном или нескольких файлах на диске и загружаются по мере необходимости.

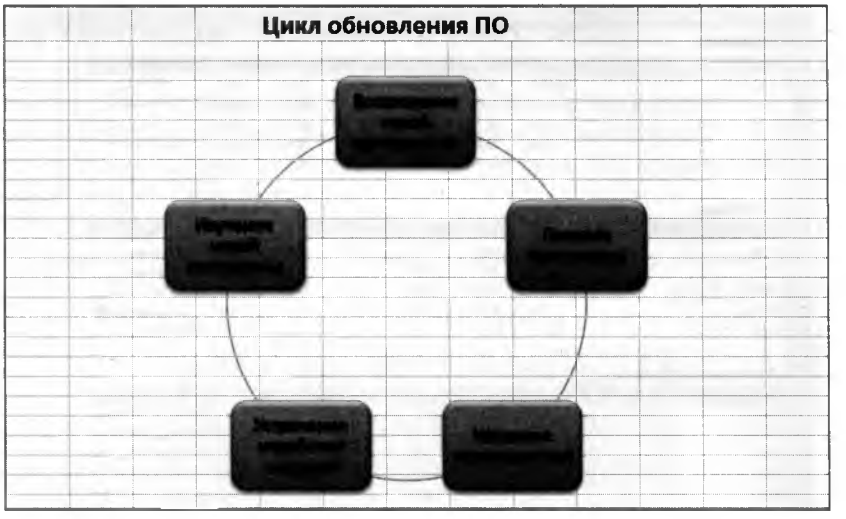


Рис. 2.23. Пример рисунка SmartArt

Базы данных рабочих листов

Обычно в качестве базы данных рабочих листов принимается прямоугольный диапазон данных, включающий заголовки столбцов.

Еще в версии Excel 2007 появилась возможность обозначить диапазон данных в качестве таблицы. Для этого достаточно выделить любую ячейку в прямоугольном диапазоне данных и выбрать команду **Вставка⇨Таблицы⇨Таблица** (Insert⇨Tables⇨Table). Благодаря таблицам обеспечивается масса преимуществ: автоматически вычисляемая нижняя строка, простая фильтрация и сортировка, автозаполнение формул в столбцах, а также упрощенное форматирование. Кроме того, если на основе таблицы создается диаграмма, она будет автоматически расширяться по мере добавления строк в таблицу.

Ряд преимуществ дает возможность работы со столбцами данных в таблице. Каждый заголовок столбца фактически является раскрывающимся списком, обеспечивающим простой доступ к возможностям сортировки и фильтрации (рис. 2.24). Строки таблицы, которые не соответствуют критериям фильтрации, временно скрываются.

Внешние базы данных

Для работы с внешними базами данных используйте команды из группы **Данные⇨Получить внешние данные** (Data⇨Get External Data). Программа Excel 2010 поддерживает работу с большим числом внешних баз данных.

Excel и Интернет

В Excel представлен набор функций, помогающих управлять ресурсами Интернета. Например, есть возможность сохранить рабочий лист или всю рабочую книгу в формате HTML, поддерживаемом веб-браузерами. Кроме того, непосредственно в ячейки можно вставлять гиперссылки, активизируемые щелчком мыши (в том числе и адреса электронной почты).

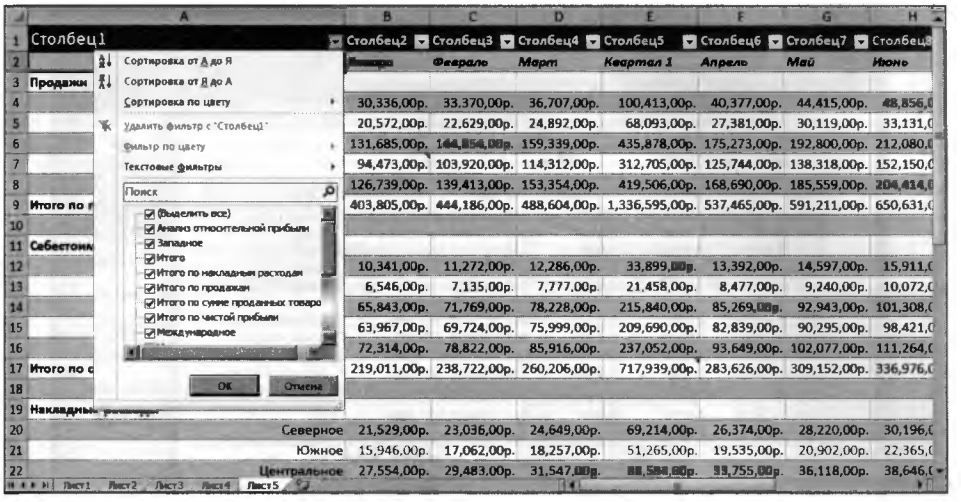


Рис. 2.24. Таблицы в Excel облегчают сортировку и фильтрацию строк

Предупреждение

В предыдущих версиях Excel формат HTML выступал в качестве *универсального файлового формата*. Другими словами, можно было сохранить рабочую книгу в формате HTML и повторно открыть ее в Excel, и ничего при этом не терялось. С появлением Excel 2007 эта возможность больше не поддерживается. Теперь HTML рассматривается исключительно в качестве формата для экспорта.

Можно также создавать веб-запросы, предназначенные для выборки данных, находящихся в локальной сети или в Интернете. Подобные запросы могут обновляться по мере загрузки на сайт новой информации. Пример такого запроса показан на рис. 2.25.

Инструменты анализа

Что касается задач анализа, то Excel справляется с ними довольно неплохо (именно по этой причине большинство пользователей обращаются к электронным таблицам). Некоторые задачи анализа решаются с помощью формул, однако Excel предлагает и другие варианты.

- **Структуры.** Режим структуры рабочей таблицы зачастую является эффективным средством управления иерархически упорядоченными данными (такими, например, как бюджет). Excel автоматически создает структуру (горизонтальную, вертикальную или смешанную), а также позволяет получать ее вручную. Единожды создав структуру, в дальнейшем можно разворачивать и сворачивать ее для отображения разных уровней детализации.
- **Analysis ToolPak (Пакет анализа).** В предыдущих версиях Excel надстройка Analysis ToolPak предназначалась для поддержки специальных инструментов анализа и функций рабочего листа, которые в основном предназначались для выполнения статистических расчетов. Начиная с версии Excel 2007 эти инструменты и функции встроены, благодаря чему Excel может применяться профессионалами в области статистики.

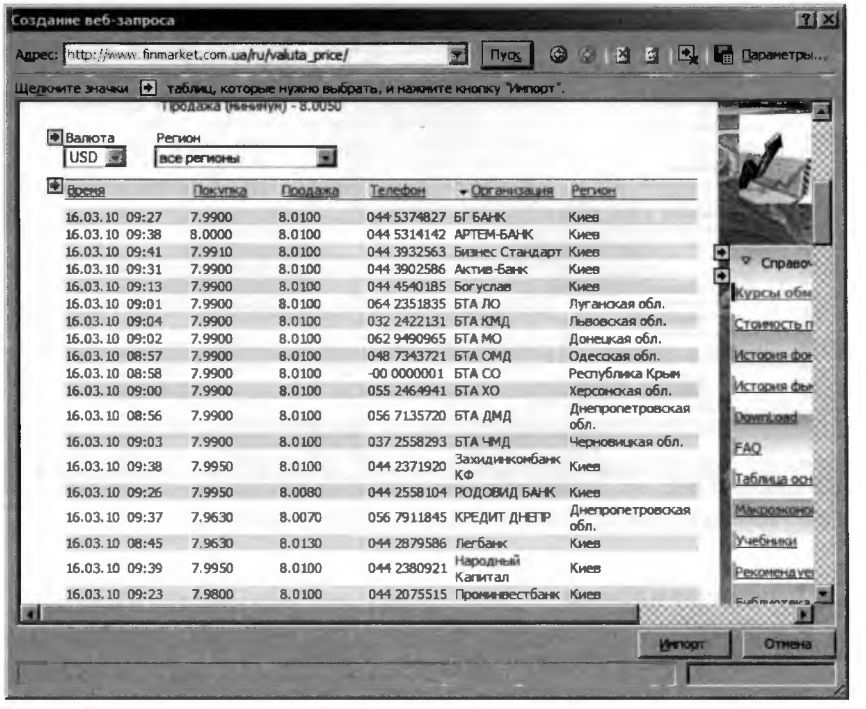


Рис. 2.25. Создание веб-запроса, предназначенного для импорта данных в рабочий лист

- **Сводные таблицы.** Одним из самых мощных инструментов Excel являются сводные таблицы. Они позволяют сводить данные в виде удобной таблицы, которую можно приспособить для решения сложных задач. Кроме того, объектами сводной таблицы можно управлять, используя возможности языка VBA. Данные этой таблицы импортируются из базы данных, которая находится на рабочем листе или загружается из внешнего хранилища, и хранятся в специальной кэш-памяти, позволяющей быстро выполнять пересчет данных после каждого изменения сводной таблицы. Пример сводной таблицы представлен на рис. 2.26.

Перекрестная ссылка

Обратитесь к главе 17 для получения сведений о работе со сводными таблицами с помощью VBA.

- **Поиск решения.** Для решения специальных линейных и нелинейных задач в Excel применяется надстройка Поиск решения (Solver), которая использует структуры "что если" для подбора данных в одних ячейках на основе ограничений, накладываемых на другие ячейки.

Новинка

В Excel 2010 модуль Поиск решения изменил свой внешний вид, а также стал более производительным.

	A	B	C	D	E	F
1	Отчет о продажах					
2						
3		Значения				
4	Названия строк	Сумма по полю Кв. 1	Сумма по полю Кв. 2	Сумма по полю Кв. 3	Сумма по полю Кв. 4	
5	⊟ Австралийская баранина	2667,6	4013,1	4836	6087,9	
6	ANTON	0	702	0	0	
7	BERGS	312	0	0	0	
8	BOLID	0	0	0	1170	
9	BOTTM	1170	0	0	0	
10	ERNSH	1123,2	0	0	2607,15	
11	GODOS	0	280,8	0	0	
12	HUNGC	62,4	0	0	0	
13	PICCO	0	1560	936	0	
14	RATTC	0	592,8	0	0	
15	REGGC	0	0	0	741	
16	SAVEA	0	0	3900	789,75	
17	SEVES	0	877,5	0	0	
18	WHITC	0	0	0	780	
19	⊟ Английский эль	551,6	665	0	890,4	
20	ANTON	0	560	0	0	
21	SAVEA	0	0	0	554,4	
22	THEBI	0	0	0	140	
23	TOMSP	179,2	105	0	0	
24	VAFFE	0	0	0	196	
25	WHITC	372,4	0	0	0	

Рис. 2.26. Сводные таблицы Excel имеют множество применений

Надстройки

Надстройкой называется программа, внедренная в Excel с целью расширения функциональных возможностей программы. Чтобы подключить надстройку, воспользуйтесь разделом Надстройки (Add-Ins) в диалоговом окне Параметры Excel (Excel Options).

Кроме надстроек, которые поставляются вместе с Excel, существуют и другие, загружаемые с веб-сайта компании Microsoft (`http://office.microsoft.com`). Более того, на рынке представлены надстройки сторонних производителей; их можно покупать или загружать из Интернета. Вы также имеете возможность создавать собственные надстройки, о чем подробно рассказывается в главе 21.

Макросы и программирование

В Excel имеются два встроенных языка программирования макросов: XLM и VBA. Изначально был создан язык XLM, но на сегодняшний день он устарел и полностью заменен на VBA. В Excel 2010 можно запускать большинство XLM-макросов, а также создавать макросы этого типа. Но при этом невозможно записывать XLM-макросы. Поэтому для создания макросов следует использовать VBA.

Перекрестная ссылка

Язык VBA подробно рассматривается в части III.

Файловые форматы

При разработке всех версий Excel во главу угла ставится совместимость между форматами файлов. В версиях от Excel 97 до Excel 2003 включительно использовался один и тот же файловый формат, поэтому проблем совместимости не возникало. С появлением Excel 2007 возник новый файловый формат, который применяется и в версии Excel 2010.

К счастью, компания Microsoft разработала "пакет совместимости", который доступен для более ранних версий Excel. Благодаря этому пакету более ранние версии Excel могут считывать и записывать файлы в новом формате XLSX.

Важно понимать разницу между совместимостью файлов и совместимостью свойств. Например, хотя пакет совместимости дает возможность открывать в Excel 2003 файлы, созданные в Excel 2007, он не заменяет свойства, внедренные в более новой версии.

Перекрестная ссылка

Обратитесь к главе 4 для получения дополнительных сведений о файловых форматах Excel, а также прочтите главу 26, в которой рассматриваются вопросы совместимости с точки зрения разработчиков.

Справочная система Excel

Один из наиболее важных компонентов Excel — справочная система. Щелкните на вопросительном знаке под строкой заголовка (либо нажмите клавишу <F1>), и появится окно справки Excel, в котором можно найти ответ на возникший вопрос или воспользоваться содержанием.

Совет

Кнопка Поиск (Search) в окне справки фактически является раскрывающимся списком. Используйте этот список для сужения диапазона поиска либо для указания источника поиска (рис. 2.27).

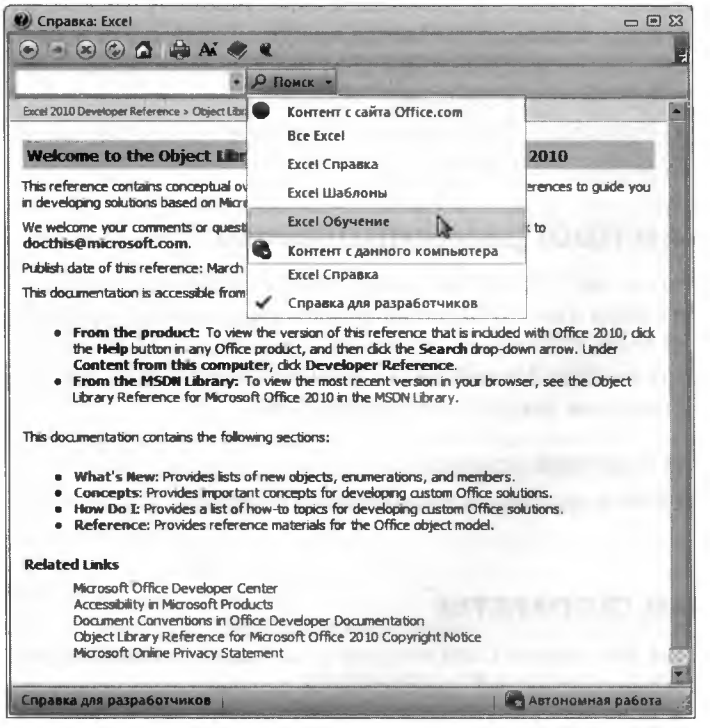


Рис. 2.27. Окно справки Excel

Глава 3

Особенности формул

В этой главе...

- О формулах
- Вычисление формул
- Ссылки на ячейки и диапазоны
- Использование имен
- Ошибки в формулах Excel
- Формулы массивов
- Подсчет и суммирование
- Работа со значениями даты и времени
- Создание мегаформул

Формулы используются во многих сложных приложениях, созданных с помощью электронных таблиц. Создание формул можно рассматривать как своего рода "программирование".

Примечание

Исчерпывающая информация о создании и применении формул приведена в моей книге *Формулы в Microsoft Excel 2010* (Диалектика, 2011).

О формулах

Формулы — это основа электронной таблицы. Если в таблице нет формул, то она представляет собой просто статический документ (который можно создать с помощью текстового процессора, обеспечивающего прекрасную поддержку статических таблиц).

Ниже приведены элементы, которые представляют часть введенной в ячейку формулы:

- операторы + (сложение) и * (умножение);
- ссылки на ячейки (в том числе имена ячеек и диапазонов);

- значения или строки;
- функции рабочих листов (например, СУММ или СРЗНАЧ).

Формула в Excel 2010 может содержать до 8192 символов. После введения формулы в ячейку последняя отображает результат выполнения формулы. Впрочем, если ячейку активизировать, то в строке формул появится сама формула. Для просмотра длинной формулы можно перетащить в вертикальном направлении границу строки формул.

Вычисление формул

Вы, возможно, заметили, что формулы в рабочем листе вычисляются сразу. И если изменить ячейку, которая используется в формуле, то результат будет пересчитан без вашего участия. Это происходит в тех случаях, когда значение параметра **Вычисления** (Excel Calculations) равно **Автоматически** (Automatic). Режим, задаваемый этим значением, установлен по умолчанию. Вычисления в рабочем листе, которые выполняются в этом режиме, характеризуются следующими правилами.

- Когда вы вносите изменения (например, вводите или редактируете данные либо формулы), Excel немедленно вычисляет значения формул уже с учетом новых или отредактированных данных.
- Иногда Excel временно приостанавливает процесс длительного вычисления формул, чтобы вы могли выполнить другие задачи, связанные с управлением рабочим листом. Когда же вы заканчиваете свои действия, вычисление возобновляется.
- Формулы вычисляются в естественном порядке. Другими словами, если формула в ячейке D12 зависит от результата вычисления формулы в ячейке D11, то сначала вычисляется значение в ячейке D11 и только затем — в ячейке D12.

Впрочем, иногда контроль над вычислением значений формул в Excel вам, возможно, придется брать на себя. Например, создавая рабочий лист с тысячами сложных формул, вы заметите, что при вычислении их значений скорость обработки данных резко снижается. В таком случае рекомендуется установить ручной режим вычислений. Для этого перейдите в группу **Формулы⇨Вычисление** (Formulas⇨Calculation) и щелкните на кнопке **Параметры вычислений** (Calculation Options).

Если при работе в ручном режиме вычислений рабочий лист содержит невычисленную формулу, то в строке состояния будет отображено сообщение **Вычислить** (Calculate). Для пересчета значений формул можно использовать следующие клавиши.

- **После нажатия клавиши <F9>** вычисляются значения формул во всех открытых рабочих книгах.
- **После нажатия комбинации клавиш <Shift+F9>** вычисляются значения формул только в активном рабочем листе. В других рабочих листах этой же рабочей книги вычисления не выполняются.
- **После нажатия комбинации клавиш <Ctrl+Shift+F9>** пересчитывается абсолютно все. Эта комбинация клавиш является незадокументированной. Используйте ее, если Excel по какой-либо причине явно рассчитывает данные неправильно или если существует необходимость пересчитать формулы, в которых применяются пользовательские функции, созданные на языке VBA.

- **После нажатия комбинации клавиш <Ctrl+Alt+Shift+F9>** происходит проверка всех зависимых формул, а также вычисляются значения во всех ячейках всех рабочих книг (включая те ячейки, для которых необходимость вычисления не указана).

Примечание

В Excel невозможно выбрать свой особый режим вычислений для отдельного рабочего листа. При изменении режима вычислений затрагиваются все открытые рабочие книги, а не только активная рабочая книга.

Ссылки на ячейки и диапазоны

В большинстве формул присутствуют ссылки не более чем на одну ячейку. Такие ссылки задаются с помощью адреса или имени (если оно задано), определяющего как ячейку, так и диапазон ячеек. Ссылки на ячейки бывают четырех типов.

- **Относительная.** Ссылка является полностью относительной. Когда формула копируется, то ссылка изменяется в соответствии с новым местоположением формулы (например, `A1`).
- **Абсолютная.** Ссылка является полностью абсолютной. Когда формула копируется, ссылка не изменяется (например, `$A$1`).
- **Абсолютная строка.** Ссылка является частично абсолютной. Когда формула копируется, та часть ссылки, которая указывает столбец, изменяется в соответствии с новым местоположением формулы, а строчная часть ссылки остается неизменной (например, `A$1`).
- **Абсолютный столбец.** Ссылка является частично абсолютной. Когда формула копируется, то строчная часть ссылки меняется в соответствии с новым местоположением формулы, а та часть ссылки, которая указывает столбец, остается неизменной (например, `$A1`).

По умолчанию все ссылки на ячейки и диапазоны являются относительными. Чтобы изменить тип ссылки, следует вручную добавить к ней знаки доллара. Можно сделать и по-другому: когда ячейка редактируется в строке формул, переместите курсор к нужному адресу, а затем нажимайте клавишу <F4> до тех пор, пока методом подбора не получите необходимый тип ссылки.

Зачем нужны неотносительные ссылки

Единственная причина, по которой когда-либо придется изменить тип ссылки, — это необходимость скопировать формулу (рис. 3.1). Пусть в ячейке C3 находится следующая формула:

```
=$B3*C$2
```

Данная формула вычисляет площадь прямоугольника для различных значений его ширины (перечисленных в столбце B) и длины (перечисленных в строке 3). Введенную формулу скопировали сначала вниз, в ячейку C7, а затем вправо, в ячейку F7. Поскольку в формуле используются ссылки (одна — с абсолютной строкой 2, другая — с абсолютным столбцом B, остальные части этих ссылок являются относительными), каждая скопированная формула все равно будет давать правильный результат. Если в формуле применяются только относительные ссылки, то в результате ее копирования все ссылки изменятся, что приведет к неправильным результатам.

	A	B	C	D	E	F	G
1			Ширина				
2			1,0	1,5	2,0	2,5	
3	Длина	1,0	1,0	1,5	2,0	2,5	
4		1,5	1,5	2,3	3,0	3,8	
5		2,0	2,0	3,0	4,0	5,0	
6		2,5	2,5	3,8	5,0	6,3	
7		3,0	3,0	4,5	6,0	7,5	
8							
9							
10							

Рис. 3.1. Пример использования неотносительных ссылок в формулах

О ссылках в стиле R1C1

Как правило, в Excel используется формат записи ссылок A1. Каждый адрес ячейки, отображаемый в таком формате, состоит из буквы, которая обозначает столбец, и числа, которое соответствует строке. Впрочем, в Excel также поддерживается формат записи ссылок R1C1. (Здесь R означает "row", т.е. "строка", а C — "column", т.е. "столбец".) В этом формате ячейка A1 обозначается как `R1C1`, а ячейка A2 — соответственно как `R2C1` и т.д.

Чтобы перейти к формату R1C1, перейдите в раздел **Формулы** (Formulas) диалогового окна **Параметры Excel** (Excel Options). Установите флажок **Стиль ссылок R1C1** (R1C1 Reference Style). После этого вы обнаружите, что все буквы столбцов заменены числами. Более того, соответствующим образом в созданных ранее формулах изменяются все ссылки на ячейки и диапазоны.

В табл. 3.1 приведены примеры формул, использующих стандартный формат записи и формат R1C1. Предполагается, что каждая из этих формул находится в ячейке B1 (также известной как `R1C2`).

Таблица 3.1. Простые формулы, выведенные в одном из двух форматов записи

Стандартный	R1C1
`=A1+1`	`=RC[-1]+1`
`=$A$1+1`	`=R1C1+1`
`=$A1+1`	`=RC1+1`
`=A$1+1`	`=R1C[-1]+1`
`=СУММ(A1:A10)`	`=СУММ(RC[-1]:R[9]C[-1])`
`=СУММ($A$1:$A$10)`	`=СУММ(R1C1:R10C1)`

Если формат записи ссылок R1C1 кажется вам весьма запутанным, то знайте, что вы не одиноки в своих умозаключениях. Он применяется для введения абсолютных ссылок, но когда используются относительные ссылки, от квадратных скобок легко сойти с ума.

Числа в квадратных скобках обозначают относительное местоположение ссылок. Например, ссылка `R[-5]C[-3]` указывает на ячейку, которая находится на пять строк выше и на три столбца левее той ячейки, в которой расположена текущая ссылка. С другой стороны, ссылка `R[5]C[3]` обозначает ячейку, которая расположена на пять строк ниже и три столбца правее текущей. Если квадратных скобок нет, то это указывает на ту

же самую строку или тот же самый столбец. Например, `R[5]C` указывает на ячейку, расположенную на пять строк ниже в текущем столбце.

Скорее всего, формат R1C1 не станет для вас используемым по умолчанию, однако он все же вам пригодится. С его помощью можно легко отыскать формулу с ошибкой. Если вы используете формат R1C1, то любые копии одной и той же формулы будут одинаковыми. Это относится ко всем типам применяемых вами ссылок на ячейки (относительных, абсолютных или смешанных). Можете перейти в режим R1C1 и проверить скопированные формулы. И если какая-либо из них отличается от остальных, то, скорее всего, она и является неправильной.

Кроме того, создавая код VBA для получения формул рабочих листов, вы, возможно, предпочтете формат R1C1.

Ссылки на другие листы или рабочие книги

Ячейки и диапазоны, на которые задаются ссылки в формуле, не обязательно должны находиться в том же листе, что и сама формула. Если в формуле требуется указать ячейку из другого листа, то перед ссылкой на саму ячейку введите имя этого листа, а после имени — восклицательный знак. Ниже приведен пример формулы со ссылкой на ячейку, расположенную в другом рабочем листе (`Лист2`).

```
=Лист2!A1+1
```

Кроме того, можно создавать формулы со ссылками на те ячейки, которые расположены в другой рабочей книге. Для этого перед ссылкой на саму ячейку введите имя рабочей книги (в квадратных скобках), имя рабочего листа и восклицательный знак.

```
=[Бюджет.xlsx]Лист1!A1
```

Если в имени рабочей книги, используемом в ссылке, содержатся пробелы, то его (вместе с именем рабочего листа) необходимо заключить в одинарные кавычки.

```
='[Бюджет на 2010 год.xlsx]Лист1'!A1
```

Указанная в ссылке рабочая книга может быть закрыта, тогда в ссылке следует указать полный путь к этой книге.

```
='C:\Бюджеты\Файлы Excel\[Бюджет на 2010 год.xlsx]Лист1'!A1
```

В формулах ссылки на рабочие книги указываются в виде пути. Однако вы вправе использовать метод указания мышью. Для этого исходный файл должен быть открытым. В данном случае создаются абсолютные ссылки на ячейки (если вы собираетесь копировать формулу в другие ячейки, то ссылки обязательно измените на относительные).

Предупреждение

Работа со ссылками может показаться сложной. Например, если для создания резервной копии исходной рабочей книги вы используете команду Файл⇨Сохранить как (File⇨Save As), то формулы со ссылками автоматически изменятся, чтобы обращаться к указанному файлу (но имеющему новое имя). Существует еще одна возможность нарушить ссылки: переименуйте исходную рабочую книгу, когда не открыта зависящая от нее рабочая книга.

Ссылки в таблицах

Начиная с версии Excel 2007 появился специальный тип диапазона, который называется таблицей (для включения таблицы в свой документ Excel воспользуйтесь командой Вставка⇨Таблицы⇨Таблица. Благодаря таблицам возможности формул существенно расширились.

При вводе формулы в ячейку таблицы Excel автоматически копирует ее во все остальные ячейки данного столбца (это происходит в том случае, если столбец пуст). Подобный столбец называется вычисляемым. Если в таблицу добавить новую строку, формула вычисляемого столбца автоматически будет вводиться в новую строку. В большинстве случаев именно это и нужно. Если же программа не должна вводить данные вместо вас, воспользуйтесь смарт-тегом, появляющимся после ввода формулы вычисляемого столбца, для отключения этого свойства.

В Excel также имеется поддержка "структурированных ссылок", с помощью которых устанавливаются ссылки на ячейки в таблице. Соответствующий пример показан на приведенном ниже рисунке (эта таблица называется `Таблица1`).

	A	B	C	D	E
1	Месяц	Штат	Доходы	Издержки	
2	Январь	Вашингтон	583	462	
3	Февраль	Вашингтон	1022	549	
4	Март	Вашингтон	881	503	
5	Январь	Орегон	764	398	
6	Февраль	Орегон	993	425	
7	Март	Орегон	882	387	
8	Итого		5505	2724	
9					
10					

Можно также создавать формулы, которые ссылаются на ячейки таблицы с помощью заголовков столбцов. В некоторых случаях это будет способствовать облегчению понимания происходящего. Однако реальное преимущество заключается в том, что формулы остаются корректными после добавления (удаления) строк таблицы. Ниже приведены примеры корректных формул.

```
=Таблица1[[#Totals],[Доходы]]
=SUM(Таблица1[Доходы])
=Таблица1[[#Totals],[Доходы]]-Таблица1[[#Totals],[Издержки]]
=SUM(Таблица1[Доходы])-СУММ(Таблица1[Издержки])
=СУММЕСЛИ(Таблица1[Штат],"Орегон",Таблица1[Доходы])
=Таблица[@Издержки]
```

В последней формуле используется символ `@`, который означает "в этой строке". Эта формула будет корректной в том случае, если одна из строк относится к таблице.

Использование имен

Одна из самых существенных возможностей программы Excel — это назначение самым разным элементам содержательных имен. Имена можно присваивать ячейкам, диапазонам ячеек, строкам, столбцам, диаграммам и другим объектам. Преимущество, которым обладает только Excel, позволяет присваивать имена тем значениям или формулам, которые даже не отображаются в ячейках рабочего листа (см. раздел "Присвоение имен константам").

Присвоение имен ячейкам и диапазонам

В Excel доступно несколько методов присвоения имен ячейкам либо диапазонам.

- Выберите команду **Формулы⇨Определенные имена⇨Присвоить имя** (Formulas⇨Named Cells⇨Name a Range) для отображения диалогового окна **Создание имени** (New Name).

- Воспользуйтесь диалоговым окном **Диспетчер имен** (Name Manager). Для его вызова выполните команду **Формулы⇨Определенные имена⇨Диспетчер имен** (Formulas⇨Defined Names⇨Name Manager) либо нажмите клавиши <Ctrl+F3>. Конечно, этот метод нельзя назвать наиболее эффективным, поскольку требуется дополнительно щелкнуть на кнопке **Создать** (New) в диалоговом окне **Диспетчер имен** (Name Manager) для отображения диалогового окна **Создание имени** (New Name).
- Выделите ячейку или диапазон, введите имя в поле **Имя** (Name) и нажмите клавишу <Enter>. По сути, поле **Имя** представляет собой раскрывающийся список, который отображается в левой части строки формул.
- Если рабочий лист включает текст, который вы хотите использовать в качестве имен смежных ячеек либо диапазонов, выделите его (вместе с именуемыми ячейками) и выполните команду **Формулы⇨Определенные имена⇨Создать из выделенного фрагмента** (Formulas⇨Defined Names⇨Create from Selection). Например, на рис. 3.2 диапазон B3:E3 называется `'Север'`, B4:E4 — `'Юг'` и т.д. В направлении по вертикали диапазон B3:B6 называется `'Квартал2'`, C3:C6 — `'Квартал2'` и т.д. Обратите внимание на то, каким образом Excel изменяет имена, чтобы сделать их корректными (использование дефиса в именах не допускается).

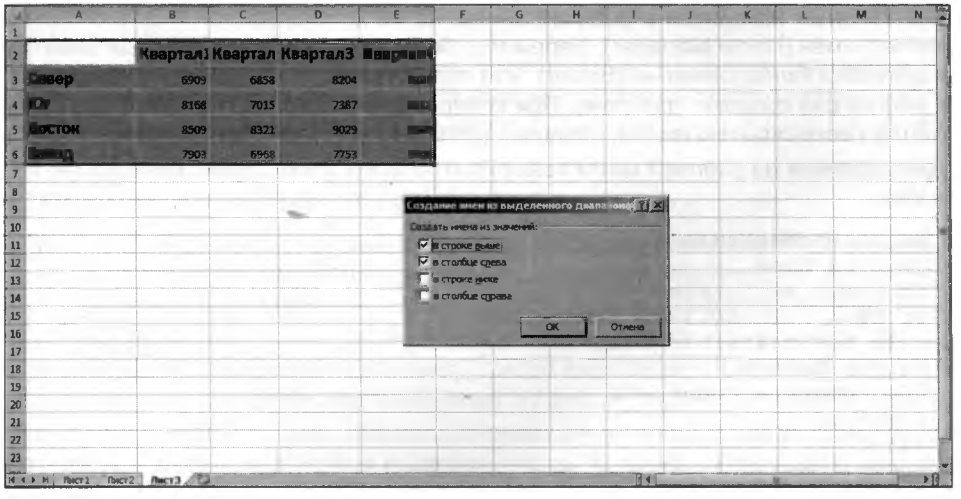


Рис. 3.2. Программа Excel позволяет создавать имена на основе описательного текста в рабочем листе

Использование имен особенно эффективно при написании кода VBA, в котором применяются ссылки на отдельные ячейки или диапазоны. Почему же так важны имена? Ответ заключается в следующем: если ячейку (или диапазон), на которую ссылается оператор VBA, переместить в другое место, то в VBA-коде эти ссылки автоматически обновляться не будут. Например, если в VBA-коде значение записывается в ячейку C4, заданную как `Range("C4")`, то после вставки новой строки над этой ячейкой или нового столбца слева от нее данные будут записываться не в требуемую ячейку. Чтобы не возникало подобных проблем, применяйте ссылки на именованные ячейки, например `Range("InterestRate")`.

Применение имен к существующим ссылкам

Когда ячейке или диапазону ячеек назначается имя, Excel не сможет автоматически использовать его вместо ссылок на ячейку или диапазон, которые уже содержатся в формулах. Предположим, в ячейке F10 находится такая формула:

```
=A1-A2
```

Если для A1 вы зададите имя `Доходы`, а для A2 — `Расходы`, то формула автоматически не будет превращена в следующую формулу:

```
=Доходы-Расходы
```

Впрочем, заменить именами ссылки на ячейки и диапазоны несложно. Вначале выделите диапазон, в котором необходимо сделать изменения. Затем выполните команду **Формулы⇨Определенные имена⇨Присвоить имя** (Formulas⇨Defined Names⇨Apply Names). В появившемся диалоговом окне выделите имена, которые следует применить при замене, и щелкните на кнопке **OK**. В результате все ссылки на ячейки и диапазоны, имеющие имена, будут заменены ссылками на имена.

Скрытые имена

Отдельные макросы и надстройки Excel создают скрытые имена. Так называются имена, которые находятся в рабочей книге, но не отображаются в диалоговом окне Диспетчер имен (Name Manager). Например, большое количество скрытых имен создает надстройка Поиск решения (Solver). Эти скрытые имена можно игнорировать. Впрочем, иногда они создают проблему. При копировании листа в другую рабочую книгу копируются также скрытые имена, создавая трудно обнаруживаемые ссылки.

Для удаления из рабочей книги всех скрытых имен используйте следующую процедуру VBA.

```
Sub DeleteHiddenNames()
   Dim n As Name
   Dim Count As Integer
   For Each n In ActiveWorkbook.Names
      If Not n.Visible Then
         n.Delete
         Count = Count + 1
      End If
   Next n
   MsgBox Count & " скрытых имен удалено."
End Sub
```

Примечание

К сожалению, не существует способа "отключения" имен. Другими словами, если в формуле используется имя, его нельзя преобразовать в явную ссылку на ячейку либо диапазон. Хуже того, если удалить имя, используемое в формуле, то программа не вернется к обычному способу адресации ячейки или диапазона ### и выведет сообщение об ошибке `#ИМЯ?`.

Воспользуйтесь разработанной мною настройкой Power Utility Pak, в состав которой входит утилита, сканирующая все формулы в выделенной области, автоматически заменяя имена адресами ячеек.

Пересечение имен

В программе Excel существует специальный *оператор пересечения*. Он вступает в действие, когда возникает необходимость в управлении несколькими диапазонами ячеек. Этим оператором является символ пробела. Используя оператор пересечения вместе с именами, можно легко создавать достаточно содержательные формулы. Для наглядного примера обратитесь к рис. 3.2. Если в ячейку ввести такую формулу:

```
=Квартал2 Юг
```

то результатом будет 7015 — пересечение диапазонов `Квартал2` и `Юг`.

Присвоение имен столбцам и строкам

Программа Excel предоставляет возможность присваивать имена отдельным строкам и столбцам. В последнем примере имя `Квартал1` присвоено диапазону B3:B6. Однако это имя можно присвоить всему столбцу B, имя `Квартал2` — столбцу C и т.д. То же самое можно сделать и по горизонтали: имя `Север` поставить в соответствие строке 3, `Юг` — строке 4 и т.д.

Оператор пересечения используется в данном случае так же, как и раньше, но теперь в рабочий лист можно добавлять другие регионы или кварталы, не изменяя при этом существующих имен.

Присваивая имена столбцам и строкам, проверяйте, чтобы в самих строках и столбцах не было лишних данных. Помните, что если вы, например, вставите значение в ячейку C7, то оно попадет в диапазон `Квартал1`.

Определение области действия

Областью действия именованной ячейки или диапазона обычно является рабочая книга. Другими словами, имя можно использовать на любом рабочем листе рабочей книги.

Существует и другой вариант — создание имен, областью действия которых является рабочий лист. Для этого перед самим именем должно стоять имя рабочего листа, а за ним — восклицательный знак (например, `Лист1!Продажи`). Если имя применяется в том листе, для которого оно предназначено, то при обращении к нему упоминание о рабочем листе можно опустить. Впрочем, обращаться можно и к тем именам уровня рабочего листа, которые определены в другом листе. В таком случае перед выбранным именем следует добавить имя рабочего листа.

Диалоговое окно **Диспетчер имен** (Name Manager) (вызывается командой **Формулы⇨Определенные имена⇨Диспетчер имен** (Formulas⇨Defined Names⇨Name Manager)) обеспечивает идентификацию имен по их области действия (рис. 3.3). Обратите внимание на возможность сортировки имен в этом диалоговом окне. Например, достаточно щелкнуть на заголовке столбца **Область** (Scope), и будет выполнена соответствующая сортировка.

Присвоение имен константам

Каждый опытный пользователь Excel знает, как создавать имена ячеек и диапазонов (хотя не все пользователи Excel применяют это на практике). Заметим, что, кроме всего прочего, имена можно применять для обращения к значениям, которые не встречаются в рабочем листе (т.е. для обращения к константам).

Предположим, в нескольких формулах рабочего листа используется конкретное значение процентной ставки. Вы можете вставить это значение в ячейку и дать ему имя (например, `Ставка`), чтобы впоследствии применять его в своих формулах. Например, обратимся к нему в следующей формуле:

```
=Ставка*A3
```

Рис. 3.3. Диспетчер имен отображает область действия для каждого определенного имени

Существует и другой способ — открыть диалоговое окно **Создание имени** (New Name) и ввести значение процентной ставки в поле **Диапазон** (Refers To) (рис. 3.4). Для открытия этого окна выполните команду **Формулы⇨Определенные имена⇨Присвоить имя** (Formulas⇨Defined Names⇨Define Name). Затем назначенное процентной ставке имя можно использовать в формулах так, как если бы оно хранилось в ячейке. В случае изменения процентной ставки всего лишь измените определение имени **Ставка** — и все ячейки, в которых оно содержится, будут обновлены.

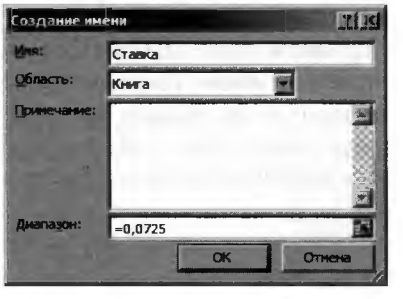


Рис. 3.4. В Excel можно присваивать имена константам, причем эти имена не отображаются в ячейках рабочего листа

Совет

Данный прием также используется для текстовых данных. Например, можно определить имя `МКП` для значения `Международная Корпорация Простаков`. После этого введите в ячейку формулу `=МКП`, и вместо нее отобразится полное название.

Присвоение имен формулам

Имена можно присваивать не только ячейкам, диапазонам и константам, но и формулам. Важно также понимать, что именованная формула, как упоминалось выше, задается не в ячейке, а в оперативной памяти. Вы также вправе ввести формулу в поле **Диапазон** (Refers To) диалогового окна **Создание имени** (New Name).

Примечание

А теперь минутку внимания. Введенная вами формула имеет относительные ссылки, если рассматривать ее с точки зрения ячейки, в которой она находится (*активной ячейки*).

На рис. 3.5 показана формула (`=A1^B1`), введенная в поле **Диапазон** (Refers To) диалогового окна **Создание имени** (New Name). В этом случае активна ячейка C1, поэтому формула обращается к двум ячейкам, которые находятся левее (обратите внимание, что ссылки на ячейки являются относительными). Если после определения имени ввести в какую-либо ячейку формулу **`=Степень`**, то значение, находящееся на две ячейки левее, будет возведено в степень, указанную в ячейке слева. Например, если в ячейках B10 и C10 находятся соответственно 3 и 4, то ввод следующей формулы в ячейку D10 приведет к выводу значения, равного 81 (3 в 4-й степени):

```
=Степень
```

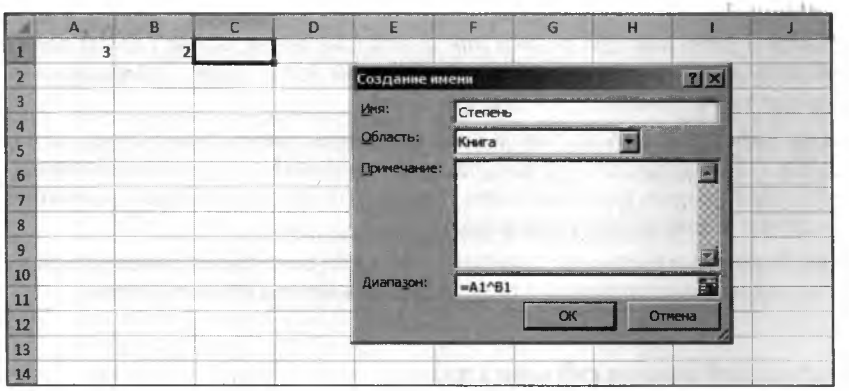


Рис. 3.5. Формуле можно присвоить имя, которое не отображается на рабочем листе

Открыв после создания именованной формулы диалоговое окно **Диспетчер имен** (Name Manager), вы обнаружите, что в столбце **Диапазон** (Refers To) отображается формула, которая является относительной для активной ячейки. Например, если активна ячейка D32, то в столбце **Диапазон** появится следующая формула:

```
=Лист1!B32^Лист1!C32
```

Обратите внимание, что в ссылки добавлено имя рабочего листа. Таким образом, если именованную формулу использовать за пределами рабочего листа, в котором она определена, то ее значения могут быть неправильными. Если же требуется применить именованную формулу в ином листе, чем `Лист1`, то из формулы придется удалить все ссылки на лист (однако сохранив восклицательные знаки). Пример приводится ниже.

```
=!A1^!B1
```

Разобравшись с именованными формулами, вы, возможно, найдете для них новое применение. Неоспоримое преимущество наблюдается в том случае, если в формулу необходимо внести изменения. Можно просто изменить определение формулы, а не редактировать каждый ее экземпляр на рабочем листе.

Компакт-диск

На прилагаемом компакт-диске содержится рабочая книга с несколькими примерами именованных формул. Эта книга называется `named formulas.xlsx`.

Совет

В диалоговом окне Создание имени (New Name) поле Диапазон (Refers To) обычно находится в режиме указателя. В этом случае облегчается ввод ссылки на диапазон — достаточно щелкнуть на рабочем листе. После нажатия клавиши <F2> происходит переключение в режим редактирования, в котором можно редактировать формулу с помощью клавиш управления курсором.

Что представляют собой имена ячеек и диапазонов

Опытные пользователи Excel часто говорят об *именованных диапазонах* и *именованных ячейках*. В данной главе мы часто употребляем эти термины. Однако эта терминология не совсем точна.

Итак, откроем секрет, чем же в действительности являются имена.

Создавая в Excel имя для ячейки или диапазона ячеек, вы на самом деле создаете именованную формулу, т.е. формулу, которой нет в ячейке. Именованные формулы находятся не в ячейках, а в буфере Excel.

Когда вы работаете в диалоговом окне Создание имени (New Name), то в поле Диапазон (Refers To) отображается формула, а в поле Имя (Name) — ее название. Примечательно, что содержимое поля Диапазон (Refers To) всегда начинается знаком равенства — он и делает содержимое поля формулой.

Если вы будете помнить этот "секрет", вам несложно будет разобраться в действиях, происходящих при создании и использовании имен в рабочих книгах.

Присвоение имен объектам

Разрешается присваивать содержательные имена не только ячейкам и диапазонам, но и таким объектам, как, например, сводные таблицы и фигуры. К ним будет проще обращаться, особенно в коде VBA.

Для изменения имени объекта, не являющегося диапазоном, воспользуйтесь полем Имя (Name), которое находится в левой части строки формул. Просто выделите объект, введите новое имя в поле Имя и нажмите клавишу <Enter>.

Примечание

Чтобы введенное вами имя не исчезло, недостаточно щелкнуть на рабочем листе после ввода имени в поле Имя. Обязательно нажмите клавишу <Enter>.

В силу некоторых соображений при работе в Excel не допускается применение поля Имя для переименования диаграммы. Для этого служит команда Работа с диаграммами⇨Макет⇨Свойства⇨Имя диаграммы (Chart Tools⇨Layout⇨Properties⇨Chart Name).

Ошибки в формулах Excel

Нередко бывает так, что, введя формулу, вы в ответ получаете значение, которое сообщает об ошибке. Формулы возвращают такое значение, если в ячейке, на которую они ссылаются, находится ошибочное значение. Это называется *цепной реакцией* — единственное ошибочное значение вызывает образование целого ряда ошибочных значений в других ячейках, в которых содержатся формулы, зависящие от ячейки с исходным значением. Инструменты, которые помогают отслеживать источники ошибок в формулах, находятся в группе Формулы⇨Зависимости формул (Formulas⇨Formula Auditing).

В табл. 3.2 перечислены значения — сообщения об ошибках, которые могут появиться в ячейках с формулами.

Таблица 3.2. Типичные ошибки в формулах Excel

Сообщение об ошибке	Описание
`#ДЕЛ/0!`	В формуле предпринимается попытка деления на нуль (эта операция запрещена законами математики). Подобное сообщение появляется и в том случае, когда в формуле осуществляется деление на содержимое пустой ячейки
`#Н/Д`	Формула ссылается (прямо или косвенно) на ячейку, в которой используется функция рабочего листа, для которой недоступны исходные данные. Кроме того, значение `#Н/Д` возвращается функцией `ПРОСМОТР`, которая не смогла найти значение
`#ИМЯ?`	В формуле используется имя, которое неизвестно программе Excel. Это происходит в том случае, когда определенное в формуле имя удалено или в тексте не совпадает количество открывающих и закрывающих кавычек
`#ПУСТО!`	В формуле применяется пересечение двух диапазонов, которые на самом деле не пересекаются (об этом рассказывалось ранее)
`#ЧИСЛО!`	Проблема связана с аргументом функции; например, если функция `КОРЕНЬ` пытается вычислить квадратный корень отрицательного числа. Это сообщение об ошибке также появляется в том случае, когда вычисляемое значение слишком велико или мало. Программа Excel не поддерживает ненулевые значения, которые по модулю меньше, чем 1E-307, либо больше, чем 1E+308
`#ССЫЛ!`	В формуле определена ссылка на недопустимую ячейку. Это может произойти в том случае, когда ячейка удалена из рабочего листа
`#ЗНАЧ!`	В формуле имеется аргумент либо операнд некорректного типа. Операнд — это значение (или ссылка на ячейку), используемое формулой для вычисления результата. Кроме того, такая ошибка появляется в том случае, когда в формуле имеется пользовательская VBA-функция, которая также содержит ошибку
`#####`	Эти символы отображаются в ячейке в следующих двух случаях: столбец недостаточно широк для отображения результата и формула возвращает отрицательные значения даты или времени

Формулы массивов

Массив — это коллекция ячеек или значений, которой управляют как единым целым. *Формулой массива* является формула специального вида, которая обрабатывает массивы данных. Результатом выполнения формулы массива может быть как единичный результат, так и набор значений, причем каждое из них помещается в отдельную ячейку (Excel допускает расположение в одной ячейке только одного значения).

Например, если вы умножаете массив размером 1×5 на массив размером 1×5, то в результате получаете массив, который имеет размер 1×5. Другими словами, результат выполнения подобной операции занимает пять ячеек рабочего листа; каждый элемент первого массива умножается на соответствующий элемент второго массива. Вы получите пять новых значений, каждое из которых будет занимать собственную ячейку. Следующая формула массива умножает значения массива A1:A5 на соответствующие значения массива B1:B5. Такая формула должна вводиться одновременно в пять ячеек.

```
{=A1:A5*B1:B5}
```

Примечание

Формула массива вводится после нажатия комбинации клавиш <Ctrl+Shift+Enter>. Для указания того, что в строке формул содержится формула массива, используются фигурные скобки, в которые она заключена. Именно таким образом будут выделяться формулы массивов, представленные в этой книге. При самостоятельном вводе формул массивов фигурные скобки не нужны.

Пример формулы массива

В Excel с помощью формул массивов можно выполнять отдельные операции над каждой ячейкой диапазона, причем во многом таким же образом, как и посредством циклических структур языка программирования. Если вам еще не приходилось использовать формулы массивов, то внимательно рассмотрите описанный далее пример.

На рис. 3.6 представлена электронная таблица с текстом в ячейках A1:A5. Цель данного упражнения состоит в том, чтобы создать единственную формулу, которая возвратит сумму, равную общему количеству символов в этом диапазоне. Если не требовать выполнения этой задачи с помощью единственной формулы, то можно создать формулу с функцией ДЛСТР, скопированной во все ячейки столбца B, а затем с помощью функции СУММ сложить результаты промежуточных формул.

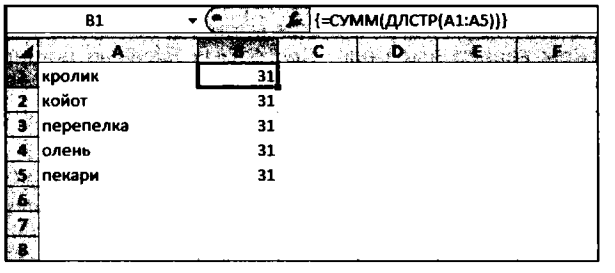


Рис. 3.6. В ячейке B1 находится формула массива, которая возвращает общее число символов, содержащихся в диапазоне A1:A5. Обратите внимание на скобки в строке формул

Для того чтобы убедиться в том, что формула массива может занимать несколько ячеек, создайте рабочий лист, показанный на рис. 3.6, а затем выполните следующие действия.

1. Выделите диапазон ячеек B1:B5.
2. Введите следующую формулу:
   `=ДЛСТР(A1:A5)`
3. Нажмите комбинацию клавиш <Ctrl+Shift+Enter>.

Эти действия выполняются для введения единственной формулы в пять ячеек. Затем введите формулу СУММ, которая складывает длины значений из ячеек B1:B5, и тогда вы увидите, что в ячейках A1:A5 находится всего 31 символ.

Главное в этом примере то, что полученные пять элементов массива в ячейках B1:B5 отображать не обязательно, так как массив может храниться в памяти. Помня об этом, вы вправе в любую пустую ячейку ввести следующую формулу (обязательно с помощью комбинации клавиш <Ctrl+Shift+Enter>):

```
{=СУММ(ДЛСТР(A1:A5))}
```

Указанная формула создает (в памяти) массив из пяти элементов, которыми являются значения длины каждой строки массива, расположенного в ячейках A1:A5. Этот массив значений используется в качестве аргумента функции СУММ — и в результате формула возвращает значение 31.

Создание календаря с помощью формулы массива

На рис. 3.7 показан рабочий лист с календарем для любого месяца. Календарь создается с помощью единственной формулы массива, которая занимает 42 ячейки.

Формула массива, введенная в ячейки B5:H10, имеет следующий вид.

```
{=ЕСЛИ(МЕСЯЦ(ДАТА(ГОД(B3);МЕСЯЦ(B3);1))<>МЕСЯЦ(ДАТА(ГОД(B3); _
  МЕСЯЦ(B3);1)-(ДЕНЬНЕД(ДАТА(ГОД(B3);МЕСЯЦ(B3);1))-1)+ _
  {0:1:2:3:4:5}*7+{1;2;3;4;5;6;7}-1);"";ДАТА(ГОД(B3); _
  МЕСЯЦ(B3);1)-(ДЕНЬНЕД(ДАТА(ГОД(B3);МЕСЯЦ(B3);1))- _
  1)+{0:1:2:3:4:5}*7+{1;2;3;4;5;6;7}-1)}
```

Формула возвращает набор последовательных числовых значений, соответствующих датам, поэтому для корректного отображения дат следует отформатировать ячейки с помощью пользовательского числового формата (“d”).

Компакт-диск

На прилагаемом компакт-диске находится рабочая книга с примером календаря, а также ряд других примеров формул массивов. Соответствующий файл называется `array examples.xlsx`. Помимо этого, на компакт-диске находится рабочая книга `yearly calendar.xlsx`, которая реализует полный календарь на весь год.

Достоинства и недостатки формул массивов

Ниже перечислены преимущества формул массивов в сравнении с формулами для одной ячейки:

- зачастую требуют меньше памяти;
- позволяют выполнять вычисления намного эффективнее;

- не требуют промежуточных формул;
- предоставляют возможность выполнять операции, которые реализовать по-другому трудно или вообще невозможною.

Рис. 3.7. С помощью единственной формулы массива можно создать календарь, отображающий любой месяц любого года

Впрочем, у формул массивов имеются и недостатки:

- некоторые формулы существенно замедляют пересчет электронной таблицы;
- мешают другим пользователям разобраться в созданной вами таблице;
- формула массива вводится с помощью специальной комбинации клавиш <Ctrl+Shift+Enter>.

Подсчет и суммирование

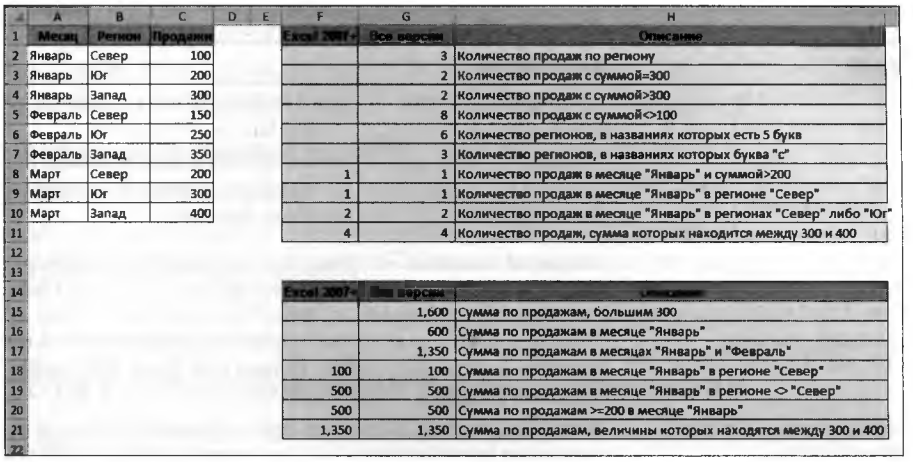
Довольно часто пользователям Excel приходится выполнять задачи, заключающиеся в условном подсчете или суммировании. В этом разделе приведен ряд примеров удобных формул, с помощью которых выполняется подсчет различных данных рабочего листа на основе одного или нескольких критериев. Эти формулы достаточно гибкие и легко могут быть адаптированы под нужды конкретного пользователя.

Примечание

В Excel 2007 появились две новые функции подсчета и суммирования, которые отсутствовали в предыдущих версиях: СЧЁТЕСЛИМН (COUNTIFS) и СУММЕСЛИМН (SUMIFS). Поэтому в книге будут представлены два варианта одних и тех же формул: версия для Excel 2007 и версия для формулы массива, которая работает во всех предыдущих версиях Excel.

На рис. 3.8 показан простой рабочий лист, демонстрирующий формулы подсчета и суммирования в действии. Определены следующие имена диапазонов:

- Месяц — A2:A10;
- Регион — B2:B10;
- Продажи — C2:C10.



	A	B	C	D	E	F	G	H
1	Месяц	Регион	Продажи			Excel 2007+	Все версии	Описание
2	Январь	Север	100				3	Количество продаж по региону
3	Январь	Юг	200				2	Количество продаж с суммой=300
4	Январь	Запад	300				2	Количество продаж с суммой>300
5	Февраль	Север	150				8	Количество продаж с суммой<>100
6	Февраль	Юг	250				6	Количество регионов, в названиях которых есть 5 букв
7	Февраль	Запад	350				3	Количество регионов, в названиях которых буква "с"
8	Март	Север	200			1	1	Количество продаж в месяце "Январь" и суммой>200
9	Март	Юг	300			1	1	Количество продаж в месяце "Январь" в регионе "Север"
10	Март	Запад	400			2	2	Количество продаж в месяце "Январь" в регионах "Север" либо "Юг"
11						4	4	Количество продаж, сумма которых находится между 300 и 400
12								
13								
14						Excel 2007+	Все версии	Описание
15							1,600	Сумма по продажам, большим 300
16							600	Сумма по продажам в месяце "Январь"
17							1,350	Сумма по продажам в месяцах "Январь" и "Февраль"
18						100	100	Сумма по продажам в месяце "Январь" в регионе "Север"
19						500	500	Сумма по продажам в месяце "Январь" в регионе <> "Север"
20						500	500	Сумма по продажам >=200 в месяце "Январь"
21						1,350	1,350	Сумма по продажам, величины которых находятся между 300 и 400
22								

Рис. 3.8. Этот простой рабочий лист демонстрирует некоторые полезные формулы подсчета и суммирования

Компакт-диск

Эта рабочая книга (включая примеры формул) доступна на прилагаемом компакт-диске (в файле `counting and summing examples.xlsx`).

Примеры формул подсчета

В табл. 3.3 представлены формулы, которые демонстрируют различные методы подсчета.

Таблица 3.3. Примеры формул подсчета

Формула	Описание
`=СЧЁТЕСЛИ(Region;"Север")`	Подсчет строк, для которых выполняется условие `Region = "Север"`
`=СЧЁТЕСЛИ(Sales;300)`	Подсчет строк, для которых выполняется условие `Sales = 300`
`=СЧЁТЕСЛИ(Sales;">300")`	Подсчет строк, для которых выполняется условие `Sales > 300`
`=СЧЁТЕСЛИ(Sales;"<>100")`	Подсчет строк, для которых выполняется условие `Sales <> 100`
`=СЧЁТЕСЛИ(Region;"?????")`	Подсчет строк, в которых название региона состоит из пяти букв
`=СЧЁТЕСЛИ(Region;"*c*")`	Подсчет строк, в которых в названии региона имеется буква 'C' (формула нечувствительна к регистру символов)
`=СЧЁТЕСЛИМН(Month; "Январь";Sales;">200")`	Подсчет строк, в которых `Month = "Январь"` и `Sales > 200` (только для Excel 2007 и более поздних версий)
`{=СУММ((Month="Январь")*(Sales>200))}`	Формула массива, которая подсчитывает строки с условием `Month = "Январь"` и `Sales > 200`
`=СЧЁТЕСЛИМН(Month; "Январь";Region;"Север")`	Подсчет строк, в которых `Month = "Январь"` и `Region = "Север"` (только для Excel 2007 и более поздних версий)

Окончание табл. 3.3

Формула	Описание
{=СУММ((Month="Январь")*(Region="Север"))}	Формула массива, которая подсчитывает строки с условием Month = "Январь" и Region = "Север"
=СЧЁТЕСЛИМН(Month;"Январь";Region;Север")+СЧЁТЕСЛИМН(Month;"Январь";Region;"Юг")	Подсчет строк, для которых выполняется условие Month = "Январь" и Region = "Север" либо "Юг" (только для Excel 2007 либо более поздних версий)
{=СУММ((Month="Январь")*((Region="Север")+(Region="Юг")))}	Формула массива, которая подсчитывает количество строк с условием Month = "Январь" и Region = "Север" либо "Юг"
=СЧЁТЕСЛИМН(Sales;">=300";Sales;"<=400")	Подсчет строк, в которых значение переменной Sales находится между 300 и 400 (только для Excel 2007 либо более поздних версий)
{=СУММ((Sales>=300)*(Sales<=400))}	Формула массива, которая подсчитывает строки, в которых значение параметра Sales находится между 300 и 400

Примеры формул суммирования

В табл. 3.4 приведены примеры формул, демонстрирующих различные методы суммирования.

Таблица 3.4. Примеры формул суммирования

Формула	Описание
=СУММЕСЛИ(Sales;">200")	Сумма по всем продажам, превышающим 200
=СУММЕСЛИ(Month;"Январь";Sales)	Сумма по всем продажам, для которых Month = "Январь"
=СУММЕСЛИ(Month;"Январь";Sales)+СУММЕСЛИ(Month;"Февраль";Sales)	Сумма по всем продажам, для которых Month = "Январь" или "Февраль"
=СУММЕСЛИМН(Продажи;Месяц;"Январь";Регион;"Север")	Сумма продаж, для которых Month = "Январь" и Регион = "Север"
=СУММЕСЛИМН(Sales;Month;"Январь";Region,"Север")	Сумма продаж, для которых Month ="Январь" и Region = "Север" (только для Excel 2007 либо более поздних версий)
{=СУММ((Month="Январь")*(Region="Север")*Sales)}	Формула массива, которая возвращает сумму продаж, для которых Month = "Январь" и Region = "Север"
=СУММЕСЛИМН(Sales;Month;"Январь";Region;"<>Север")	Сумма продаж, для которых Month = "Январь" и Region <> "Север" (только для Excel 2007 либо более поздних версий)
{=СУММ((Month="Январь")*(Region<>"Север")*Sales)}	Формула массива, которая возвращает сумму продаж, для которых Month = "Январь" и Region <> "Север"
=СУММЕСЛИМН(Sales;Month;"Январь";Sales;">=200")	Сумма продаж, для которых Month = "Январь" и Sales >= 200 (только для Excel 2007 либо более поздних версий)
{=СУММ((Month="Январь")*(Sales>=200)*(Sales))}	Формула массива, которая возвращает сумму продаж, для которых Month = "Январь" и Sales >= 200
=СУММЕСЛИМН(Sales;Sales;">=300";Sales;"<=400")	Сумма продаж, величины которых находятся между 300 и 400 (только для Excel 2007 либо более поздних версий)
{=СУММ((Sales>=300)*(Sales<=400)*(Sales))}	Формула массива, возвращающая сумму продаж, величины которых находятся между 300 и 400

Другие инструменты подсчета

Ниже перечислены дополнительные способы подсчета количества или суммирования содержимого ячеек, удовлетворяющих определенным критериям:

- фильтрация (с помощью таблиц);
- расширенная фильтрация;
- применение функций БСЧЁТ (DCOUNT) и БДСУММ (DSUM);
- работа со сводными таблицами.

Для получения дополнительных сведений обратитесь к справочной системе.

Работа со значениями даты и времени

Для хранения значений даты в Excel применяется система последовательной нумерации. Самой ранней датой, которую "понимает" программа Excel, является 1 января 1900 года. Этой дате соответствует число 1. Дата "2 января 1900 года" равна следующему значению числовой последовательности — 2 и т.д.

Вам не придется анализировать, каким же числом представлена интересующая вас дата. Достаточно ввести дату в привычном формате, а Excel позаботится о ее корректной обработке. Например, если требуется задать 15 августа 2010 года, то просто введите **15 августа 2010 г** (или используйте другой стандартный формат представления даты). Программа Excel интерпретирует введенные данные и сохранит их в виде значения 40405, которое и является числовым значением для указанной даты.

Примечание

В этой главе даты представлены в формате, характерном для русской версии Excel 2010. Он несколько отличается от принятого в США (и в английской версии Excel) формата представления дат. Например, в США эта же дата записывается в виде **August 15, 2010**.

Ввод значений даты и времени

Работая со значениями времени, вы вводите в ячейку одно из них, пользуясь для этого одним из стандартных форматов. Система представления даты, применяемая в Excel, заключается в расширенном форматировании. При этом в него добавляется дробная часть, которая обозначает долю суток, отмеренную введенным временем. Другими словами, Excel представляет время, пользуясь для этого той же системой, что и при представлении дат, независимо от того, в каких единицах измеряется это время: в часах, минутах или секундах. Например, числовое значение даты "15 августа 2010 года" — 40405. А вот полдень (середина суток) представлено значением 40405,5. Заметьте, что вам не потребуется вводить дробные числовые значения для определенного времени суток.

Поскольку дата и время хранятся в виде числовых значений, над ними допускается выполнять любые вычисления. Например, можно ввести формулу для подсчета количества дней между двумя датами.

```
=A2-A1
```

Совет

Когда дело доходит до подсчета времени, ситуация усложняется. Если время вводится без даты, в качестве даты берется 0 января 1900 года (соответствует значению 0). Это не проблема, если только в результате подсчетов не получится отрицательное значение времени. В подобном случае Excel выведет сообщение об ошибке (в виде #########). Что делать в такой ситуации? Перейти к формату дат 1904 года. Перейдите в диалоговое окно Параметры Excel (Excel Options), выберите раздел Дополнительно (Advanced) и установите флажок Использовать систему дат 1904 (Use 1904 Date System). Не забывайте, что переход к этой системе может привести к неадекватному толкованию дат, ранее введенных в текущий рабочий лист или в связанные рабочие книги.

Совет

Иногда значения времени используются для представления длительности времени. Например, может возникнуть необходимость подсчитать количество рабочих часов в неделе. При складывании значений времени обнаруживается, что нельзя получить значение, которое больше 24 часов. Для каждого 24-часового периода Excel добавляет к дате еще один день. Решить данную проблему можно путем изменения числового формата — выделите квадратными скобками ту часть значения, в которой указано количество часов. Например, ниже представлен числовой формат, в котором может отображаться более 24-х часов.

```
[чч]:мм
```

Использование дат до 1900 года

Как вы понимаете, мир начал свое существование не с 1 января 1900 года. И тем, кто использует Excel для работы с исторической информацией, часто приходится иметь дело с датами, намного более ранними, чем 1 января 1900 года. К сожалению, для управления такими датами подходит только один способ — их необходимо вводить в ячейку, как текст. Например, Excel не будет против, если в ячейку вы введете дату **4 июля 1776 года**.

Впрочем, над датами, введенными, как текст, нельзя выполнять никаких операций. Например, изменить формат даты вы не сможете, как не сможете определить день недели, на который эта дата приходится, а также подсчитать дату, отстоящую от текущей на семь дней.

Но все не так уж и печально, поскольку в VBA поддерживается расширенный набор дат. Я воспользовался рядом функций рабочего листа VBA, которые могут работать с датами до 1900 года. Результаты применения этих функций показаны на рис. 3.9. Перед вами превосходный пример расширения возможностей Excel с помощью VBA.

Перекрестная ссылка

Дополнительные сведения по работе с дополнительными функциями дат можно найти в главе 10.

Создание мегаформул

Зачастую для получения необходимого результата в электронных таблицах используются промежуточные формулы, т.е. формула может зависеть от других формул, которые, в свою очередь, зависят от еще каких-либо формул. Добившись, чтобы все они ра-

ботали правильно, вы получите возможность удалить промежуточные формулы и применить вместо них единственную формулу, которая называется *мегаформулой*. Каковы ее преимущества? Используется меньше ячеек (и, следовательно, рабочий лист не так загроможден), а также данные пересчитываются быстрее. Основной недостаток заключается в том, что такую формулу совершенно невозможно понять или изменить.

	A	B	C	D	E	F	G	H	I
1	*Эта рабочая книга включает восемь функций VBA,*								
2	*которые могут работать с датами до 1900 года.*								
3									
4	**Примеры: дни рождения президентов США**								
5									
6	**Президент**	**Год**	**Месяц**	**День**	**День рождения**	**Кол-во дней**	**Кол-во лет**	**День недели**	
7	Джордж Вашингтон	1732	2	22	22 февраля, 1732	101,678	278	Пятница	
8	Джон Адамс	1735	10	30	30 октября, 1735	100,332	274	Воскресенье	
9	Томас Джефферсон	1743	4	13	13 апреля, 1743	97,610	267	Суббота	
10	Джеймс Мэдисон	1751	3	16	16 марта, 1751	94,716	259	Вторник	
11	Джеймс Монро	1758	4	28	28 апреля, 1758	92,116	252	Пятница	
12	Джон Куинси Адамс	1767	7	11	11 июля, 1767	88,755	243	Суббота	
13	Эндрю Джексон	1767	3	15	15 марта, 1767	88,873	243	Воскресенье	
14	Мартин Ван Бюрен	1782	12	5	5 декабря, 1782	83,129	227	Четверг	
15	Уильям Генри Гаррисон	1773	2	9	9 февраля, 1773	86,715	237	Вторник	
16	Джон Тайлер	1790	3	29	29 марта, 1790	80,458	220	Понедельник	
17	Джеймс Нолк Полк	1795	11	2	2 ноября, 1795	78,414	214	Понедельник	
18	Закари Тейлор	1784	11	24	24 ноября, 1784	82,409	225	Среда	
19	Миллорд Филмор	1800	1	7	7 января, 1800	76,887	210	Вторник	
20	Франклин Пирс	1804	11	23	23 ноября, 1804	75,106	205	Пятница	
21	Джеймс Бьюкенен	1791	4	23	23 апреля, 1791	80,068	219	Суббота	
22	Авраам Линкольн	1809	2	12	12 февраля, 1809	73,564	201	Воскресенье	
23	Эндрю Джонсон	1808	12	29	29 декабря, 1808	73,609	201	Четверг	
24	Улисс Симпсон Грант	1822	4	27	27 апреля, 1822	68,742	188	Суббота	

Рис. 3.9. Дополнительные функции для работы с датами позволяют работать с датами до 1900 года

Рассмотрим пример. Представьте себе рабочий лист со столбцом, в котором перечислены имена (и фамилии) людей. Предположим, что из этих имен с фамилиями следует убрать все вторые имена и инициалы. Однако не у всех людей в списке есть такие имена и инициалы. На редактирование ячеек вручную уйдут многие часы работы, причем даже команда Excel Данные⇨Работа с данными⇨Текст по столбцам (Data⇨Data Tools⇨Convert Text to Table) не особо поможет. Поэтому необходимо прибегнуть к помощи формул. Данная задача не так уж и трудна, но для ее решения обычно требуется использовать несколько промежуточных формул.

На рис. 3.10 представлен результат довольно удачного решения — применено шесть промежуточных формул, перечисленных в табл. 3.5. Имена с фамилиями находятся в столбце A, а конечный результат — в столбце H. Что же касается столбцов B–G, то в них как раз и содержатся промежуточные формулы.

	A	B	C	D	E	F	G	H	I
1	Имя	Формула 1	Формула 2	Формула 3	Формула 4	Формула 5	Формула 6	Результат	
2	Иван Иванов	Иван Иванов	5	#ЗНАЧ!	5	Иван	Иванов	Иван Иванов	
3	Степан П. Степанов	Степан П. Степанов	7	10	10	Степан	Степанов	Степан Степанов	
4	Рэм Ильич Иванов	Рэм Ильич Иванов	4	10	10	Рэм	Иванов	Рэм Иванов	
5	Сэм Иванович Иванов	Сэм Иванович Иванов	4	13	13	Сэм	Иванов	Сэм Иванов	
6	Джон И. Бартер	Джон И. Бартер	5	8	8	Джон	Бартер	Джон Бартер	
7	Р.Дж Смит	Р.Дж Смит	5	#ЗНАЧ!	5	Р.Дж	Смит	Р.Дж Смит	
8	Р. Джей Смит	Р. Джей Смит	3	8	8	Р.	Смит	Р. Смит	
9	Тим Джонс	Тим Джонс	4	#ЗНАЧ!	4	Тим	Джонс	Тим Джонс	
10									
11									

Рис. 3.10. Для удаления отчества и инициалов потребуется шесть промежуточных формул

Таблица 3.5. Промежуточные формулы, введенные в столбцах B:G

Столбец	Промежуточная формула	Выполняемые функции
B	=СЖПРОБЕЛЫ(A2)	Удаляет лишние пробелы
C	=НАЙТИ(" ",B2,1)	Находит первый пробел
D	=НАЙТИ(" ",B2,C2+1)	Находит второй пробел. Возвращает значение #ЗНАЧ! в случае отсутствия второго пробела
E	=ЕСЛИ(ЕОШИБКА(D2),C2,D2)	Использует первый найденный пробел, если не найден второй
F	=ЛЕВСИМВ(B2,C2)	Выделяет имя
G	=ПРАВСИМВ(B2,LEN(B2)-E2)	Выделяет фамилию
H	=F2&G2	Объединяет два имени

Можно избавиться от всех промежуточных формул, создав мегаформулу. Для этого выполните следующее: создав промежуточные формулы, перейдите к конечной результирующей формуле и замените в ней каждую ссылку на ту или иную ячейку копией формулы, которая в этой ячейке находится (копия вводится без знака равенства). К счастью, для копирования и вставки можно использовать буфер обмена. Повторяйте эти действия до тех пор, пока в ячейке H2 не исчезнут все ссылки, кроме ссылок на ячейку A2. В конечном итоге у вас в одной ячейке получится следующая мегаформула.

```
=ЛЕВСИМВ(СЖПРОБЕЛЫ(A2),НАЙТИ _
   (" ",СЖПРОБЕЛЫ(A2),1))&ПРАВСИМВ(СЖПРОБЕЛЫ(A2),ДЛСТР( _
   СЖПРОБЕЛЫ(A2))- ЕСЛИ(ЕОШИБКА(НАЙТИ(" ",СЖПРОБЕЛЫ(A2), _
   НАЙТИ(" ",СЖПРОБЕЛЫ(A2),1)+1)), НАЙТИ(" _
   ",СЖПРОБЕЛЫ(A2),1),НАЙТИ(" ",СЖПРОБЕЛЫ(A2),НАЙТИ _
   (" ",СЖПРОБЕЛЫ(A2),1)+1)))
```

Убедившись, что мегаформула работает, можете удалить столбцы с промежуточными формулами, поскольку последние вам больше не понадобятся.

Мегаформула выполняет те же самые задачи, что и все промежуточные формулы, однако понять, как она работает, может только ее автор. Если вы собираетесь использовать мегаформулы, то перед их созданием проверьте, правильно ли работают промежуточные формулы. Или еще лучше — сохраните отдельно копию этих формул (на тот случай, если в расчетах обнаружится ошибка или в их алгоритм потребуется внести изменения).

Еще один способ решения этой проблемы заключается в создании пользовательской VBA-функции рабочего листа. Затем мегаформула заменяется следующей простой формулой:

```
=NOMIDDLE(A1)
```

Фактически я создал функцию, которая сравнима с промежуточными формулами и мегаформулой. Ниже приводится ее листинг.

```
Function NOMIDDLE(n) As String
   Dim FirstName As String, LastName As String
   n = Application.WorksheetFunction.Trim(n)
   FirstName = Left(n, InStr(1, n, " "))
   LastName = Right(n, Len(n) - InStrRev(n, " "))
   NOMIDDLE = FirstName & LastName
End Function
```

Компакт-диск

Рабочая книга, которая содержит промежуточные формулы, мегаформулу и VBA-функцию NOMIDDLE, находится на прилагаемом компакт-диске. Соответствующий файл называется megaformula.xlsm.

Сложность мегаформул наталкивает на мысль, что при их использовании вычисления могут существенно замедляться. На самом деле это не так. Создадим рабочий лист, в котором мегаформула использовалась 175000 раз. Затем создадим другой рабочий лист, в котором применено шесть промежуточных формул. Вы увидите, что мегаформула работает существенно быстрее и создает файл намного меньшего размера (табл. 3.6).

Таблица 3.6. Сравнение быстродействия в случае применения мегаформул и промежуточных формул

Метод	Продолжительность вычисления, с	Размер файла, Мбайт
Промежуточные формулы	5,8	12,6
Мегаформула	3,9	2,95

Фактические результаты зависят от скорости системы, количества оперативной памяти, а также обрабатываемой формулы.

Функция VBA выполняется намного медленнее — автор прервал ее выполнение после 5 минут тестирования. Как правило, VBA-функции выполняются медленнее встроенных функций Excel.

Глава 4

Файлы Excel

В этой главе...

- Запуск Excel
- Типы файлов
- Работа с файлами шаблонов
- Содержимое файла Excel
- Файл OfficeUI
- Файл XLB
- Файлы надстроек
- Настройки Excel в системном реестре

А теперь рассмотрим, как устроены файлы Excel, и начнем с ответа на вопрос "Каким же образом можно запустить саму программу Excel?"

Запуск Excel

Программа Excel может запускаться по-разному в зависимости от способов ее установки: можно щелкнуть на значке, находящемся на рабочем столе, воспользоваться кнопкой Пуск (Start) либо дважды щелкнуть мышью на значке файла, связанного с приложением Excel. Независимо от выбранного метода происходит запуск исполняемого файла `excel.exe`.

После запуска Excel происходит следующее:

- считываются настройки, находящиеся в системном реестре Windows;
- считываются и применяются настройки панели быстрого доступа либо ленты, находящиеся в файле `Excel.officeUI`;
- открывается файл настройки меню/панели инструментов с расширением `*.xlb`;
- открываются все установленные надстройки (выбраны в диалоговом окне Надстройки (Add-Ins));

- открываются все рабочие книги, находящиеся в каталоге XLStart;
- открываются все рабочие книги, находящиеся в альтернативном каталоге автозапуска (указывается в разделе Дополнительно (Advanced) в диалоговом окне Параметры Excel (Excel Options));
- определяется наличие сбоя при последнем запуске Excel (в этом случае отображается перечень автоматически восстановленных рабочих книг);
- отображается пустая рабочая книга, если только пользователь не указал открываемую рабочую книгу либо один или более файлов находятся в каталоге XLStart или в альтернативном начальном каталоге.

Программа Excel может быть установлена в любом месте. Но в большинстве случаев исполняемый файл Excel находится в каталоге установки, заданном по умолчанию:

```
C:\Program Files\Microsoft Office\Office14\EXCEL.EXE
```

Можно создать один или более ярлыков, соответствующих этому выполняемому файлу, причем сами ярлыки могут быть настроены путем изменения различных параметров, которые также называются переключателями командной строки. Эти переключатели перечислены в табл. 4.1.

Таблица 4.1. Переключатели командной строки Excel

Переключатель	Выполняемая функция
имя_файла	Открытие указанного файла. Параметр *имя_файла* не требует переключателя командной строки
/r *имя_файла*	Открытие указанного файла в режиме чтения
/t *имя_файла*	Открытие указанного файла в виде шаблона
/n *имя_файла*	Открытие указанного файла в виде шаблона (аналогичен предыдущему переключателю)
/e	Запуск Excel без создания рабочей книги и без отображения всплывающего экрана
/p *каталог*	Выбор в качестве активного пути каталога, который отличается от каталога, заданного по умолчанию
/s	Запуск Excel в безопасном режиме, который не предусматривает загрузку надстроек или файлов, находящихся в каталоге XLStart или в альтернативных каталогах автозапуска
/m	Программа создает рабочую книгу с единственным макролистом Microsoft Excel 4.0 (устаревший объект)

Можно поэкспериментировать с этими переключателями в окне, открывающемся после выполнения команды Пуск⇨Выполнить (Start⇨Run). Заключите путь, ведущий к выполняемому файлу Excel, в кавычки, и введите пробел и переключатель командной строки. Соответствующий пример показан на рис. 4.1.

Один из способов выбора переключателя командной строки заключается в редактировании свойств ярлыка, используемого для запуска Excel. Например, если у вас есть время и желание изменить каталог автозапуска на c:\xlfiles, можете настроить ярлык Windows. В этом случае следует указать переключатель /p, а также задать имя папки.

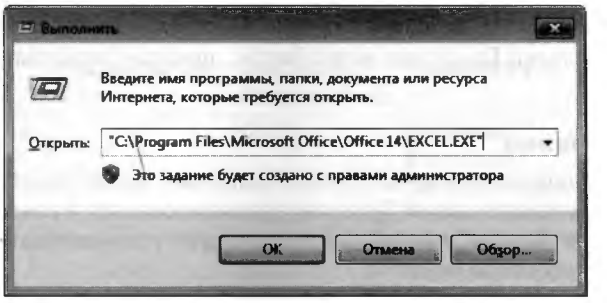


Рис. 4.1. Запуск Excel из диалогового окна Выполнить (Run)

Примечание

Приведенные ниже инструкции относятся к Windows 7.

Начнем со значка, используемого для запуска Excel. Щелкните на нем правой кнопкой мыши и в контекстном меню выберите пункт **Свойства** (Properties). Находясь в диалоговом окне **Свойства ярлыка** (Shortcut Properties), выберите вкладку **Ярлык** (Shortcut) и в поле **Объект** (Target) (рис. 4.2) введите следующую информацию:

```
"C:\Program Files\Microsoft Office\Office14\EXCEL.EXE" /p c:\xlfiles
```

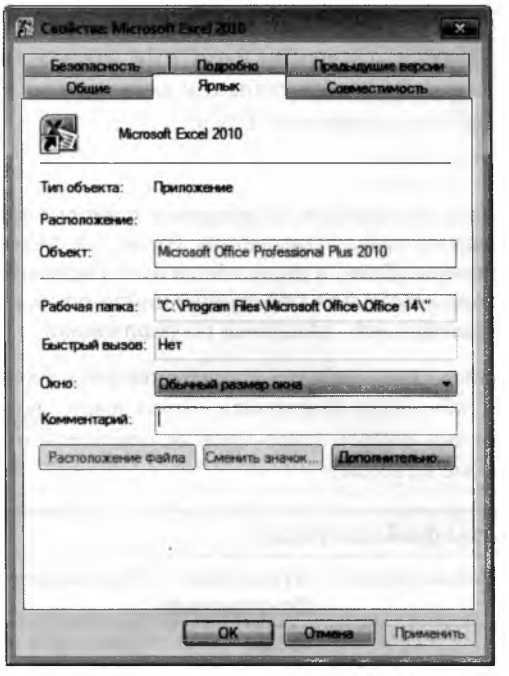


Рис. 4.2. Настройка свойств ярлыка, применяемого для запуска Excel

Имейте в виду, что путь, ведущий к файлу `excel.exe`, может отличаться в зависимости от выбранного способа установки и версии Excel.

Можно также определить клавишу быстрого доступа, с помощью которой запускается Excel. Если программа Excel уже выполняется, нажатие этой клавиши приведет к ее активизации.

Примечание

Можно одновременно выполнять несколько копий Excel на одной системе. При этом каждая копия будет интерпретироваться в качестве отдельной задачи. Для беспроблемной работы следует устанавливать различные копии Excel в порядке их выпуска.

Типы файлов

По умолчанию в Excel 2010 используются файлы рабочих книг XLSX, но они не являются единственными. В следующем разделе приводится обзор типов файлов, с которыми может работать Excel 2010.

Примечание

Начиная с версии Excel 2007 компания Microsoft не поддерживает форматы файлов электронных таблиц Lotus и Quattro Pro.

Форматы файлов Excel

С появлением Excel 2007 в обиход вошел новый стандартный файловый формат, который продолжает применяться и в версии Excel 2010. Несмотря на вышесказанное две новейшие версии Excel не утратили способности считывать и изменять файлы, сохраненные в более старых файловых форматах Excel.

Совет

Для изменения параметров сохранения файлов, заданных по умолчанию, выберите команду Файл⇨Параметры Excel (File⇨Excel Options) и выберите раздел Сохранение (Save) в диалоговом окне Параметры Excel (Excel Options). В этом разделе находится раскрывающийся список, в котором можно выбрать файловый формат, заданный по умолчанию.

В табл. 4.2 перечислены типы файлов, поддерживаемых Excel 2010. Имейте в виду, что рабочая книга Excel либо файл надстройки может иметь любое расширение. Другими словами, при сохранении файлов этого типа не обязательно использовать стандартные расширения, указанные в таблице.

Таблица 4.2. Форматы файлов Excel

Тип файла	Расширение	Открытие/ Сохранение	Примечание
Рабочая книга Excel	`xlsx`	Да/Да	Файловый формат Excel 2010, заданный по умолчанию. Не предусматривает сохранение макросов VBA или XLM
Рабочая книга Excel с поддержкой макросов	`xlsm`	Да/Да	Файловый формат Excel 2010, применяемый для сохранения рабочих книг с макросами

Окончание табл. 4.2

Тип файла	Расширение	Открытие/ Сохранение	Примечание
Двоичная рабочая книга Excel	xlsb	Да/Да	Двоичный файловый формат Excel 2010 (BIFF12). Представляет собой обновленную версию предыдущего формата XLS (BIFF8)
Шаблоны	xltx	Да/Да	Файловый формат Excel 2010, предназначенный для хранения шаблонов. Не поддерживает хранение макрокода VBA либо XLM
Шаблоны с поддержкой макросов	xltm	Да/Да	Файловый формат Excel 2010, предназначенный для хранения шаблонов с макросами
Надстройки Excel	xlam	Да/Да	Файловый формат Excel 2010, предназначенный для хранения надстроек. В этом файле могут также храниться макросы VBA или XLM
Рабочая книга Excel 97–Excel 2003	xls	Да/Да	Двоичный файловый формат Excel (BIFF8), который совместим с рабочими книгами от Excel 97 до Excel 2003
Шаблоны Excel 97–Excel 2003	xlt	Да/Да	Двоичный формат шаблонов Excel (BIFF8), который совместим с шаблонами от Excel 97 до Excel 2003
Надстройки Excel 97–Excel 2003	xla	Да/Да	Двоичный файловый формат Excel (BIFF8), предназначенный для хранения настроек, совместимых с Excel 97–Excel 2003
Рабочая книга Microsoft Excel 5.0/95	xls	Да/Да	Двоичный файловый формат Excel (BIFF5), совместимый с Excel 5.0 и Excel 95
Электронная XML-таблица, совместимая с Excel 2003	xml	Да/Да	Файловый формат Microsoft XML Spreadsheet 2003 (XMLSS)

Примечание

Пользователи Microsoft Office XP и Office 2003 могут установить пакет Microsoft Office Compatibility Pack, с помощью которого можно открывать и сохранять документы с использованием файловых форматов Office 2007 и Office 2010. Этот пакет доступен на веб-сайте `http://office.microsoft.com`.

Форматы текстовых файлов

При открытии текстового файла в Excel появится диалоговое окно мастера импорта текста (Text Import Wizard), которое призвано помочь пользователю в нелегком деле преобразования формата текстового файла.

Совет

Если помощь мастера импорта текста не требуется, во время щелчка мышью на кнопке OK нажмите клавишу <Shift>.

В табл. 4.3 перечислены типы текстовых файлов, которые поддерживаются в Excel 2010. Для всех текстовых файлов имеет место ограничение — единственный рабочий лист.

Таблица 4.3. Типы текстовых файлов

Тип файла	Расширение	Открытие/ Сохранение	Примечание
CSV (значения, разделенные запятыми)	csv	Да/Да	Столбцы визуально разделены запятыми, а строки — символами возврата каретки. В Excel также поддерживаются подтипы этого формата для Macintosh и MS-DOS
Отформатированный текст	prn	Да/Да	Столбцы разделены пробелами, а строки — символами возврата каретки
Текст	txt	Да/Да	Столбцы разделены символом табуляции, а строки — символом возврата каретки. В Excel поддерживаются подтипы этого формата для Macintosh, MS-DOS и Unicode
Формат обмена данными (Data Interchange Format — DIF)	dif	Да/Да	Формат обмена данными (Data Interchange Format — DIF)
Символическая ссылка (Symbolic Link — SYLK)	slk	Да/Да	Символическая ссылка (Symbolic Link — SYLK)

Форматы файлов баз данных

В табл. 4.4 перечислены типы баз данных, которые поддерживаются Excel 2010. Для всех файлов баз данных имеет место ограничение — единственный рабочий лист.

Таблица 4.4. Типы файлов баз данных

Тип файла	Расширение	Открытие/ Сохранение	Примечание
Access	mdb, mde, accdb, accde	Да/Нет	Можно открыть одну таблицу из базы данных
dBase	dbf	Да/Нет	Этот файловый формат был разработан компанией Ashton-Tate
Другие базы данных	Различные	Да/Нет	Используя команды из группы Данные⇨ Получить внешние данные (Data⇨Get External Data), можно импортировать данные из различных источников данных. При этом используются текущие подключения к этим источникам либо запросы, определенные в вашей системе

Другие форматы файлов

В табл. 4.5 перечислены другие форматы файлов, поддерживаемые в Excel 2010.

Таблица 4.5. Другие форматы файлов

Тип файла	Расширение	Открытие/ Сохранение	Примечание
Язык гипертекстовой разметки (Hypertext Markup Language — HTML)	`htm`, `html`	Да/Да	Начиная с версии Excel 2007 отсутствует непосредственная поддержка HTML-файлов. Если сохранить подобный файл в Excel, а затем попытаться его открыть, возможна потеря данных
Веб-страница в одном файле (Single File Web Page)	`mht`, `mhtml`	Да/Да	Также известен как архивированная веб-страница (Archived Web Page). Единственные браузеры, которые могут отображать эти файлы, — Microsoft Internet Explorer и Opera
Электронная таблица OpenDocument	`ods`	Да/Да	Файловый формат, разработанный компаниями Sun Microsystems и OASIS. Может считываться процессорами электронных таблиц с открытым кодом, такими как OpenOffice
Формат переносимого документа (Portable Document Format — PDF)	`pdf`	Нет/Да	Этот файловый формат разработан фирмой Adobe
XML-спецификация формата PDF (XML Paper Specification PDF)	`xps`	Нет/Да	Альтернативный PDF формат, разработанный компанией Microsoft

Файлы рабочих областей

Файл рабочей области — это специальный тип файла, в котором содержатся сведения о рабочей области Excel. Например, если ваш проект содержит две рабочие книги, причем окна этих книг нужно расположить определенным образом, опишите конфигурацию этих окон в XLW-файле. После этого, независимо от того, будете ли вы открывать XLW-файл, Excel настраивается на выбранную рабочую область.

Для сохранения файла рабочей области выберите команду Вид⇨Окно⇨Сохранить рабочую область (View⇨Window⇨Save Workspace) и после отображения соответствующего запроса введите имя файла.

Для открытия файла рабочей области воспользуйтесь командой Файл⇨Открыть (File⇨Open) и выберите файл рабочей области (файлы с расширением `xlw`) в раскрывающемся списке Тип файлов (Files of Type).

Учтите, что файл рабочей области не содержит каких-либо рабочих книг —только конфигурационная информация, которая приводит к отображению этих рабочих книг в вашей рабочей области Excel. Поэтому, если вы намереваетесь передать кому-либо файл рабочей области, не забудьте наравне с файлами XLW включить файлы необходимых рабочих книг.

Работа с файлами шаблонов

Шаблон — это, по сути, модель, которая служит основой для выполнения дальнейших действий. Шаблон Excel — это рабочая книга, которая применяется для создания других рабочих книг. В качестве шаблона можно использовать любую рабочую книгу (сохранив ее с расширением XLTX). Подобная практика применяется в том случае, когда приходится регулярно создавать похожие файлы. Например, предположим, что ежемесячно формируется отчет по продажам. В этом случае стоит потратить немного времени и создать шаблон, который содержит все формулы и диаграммы, используемые в отчете. Теперь, после создания файла на основе этого шаблона, достаточно будет ввести требуемые значения.

Просмотр шаблонов

Программа Excel 2010 предоставляет своим пользователям доступ к целому набору шаблонов. Для просмотра шаблонов Excel выберите команду **Файл⇨Создать** (File⇨New), после чего на экране появится диалоговое окно **Доступные шаблоны** (Available Templates).

В разделе **Шаблоны Office.com** (Office Online Templates) находится множество категорий шаблонов. После щелчка на категории перед вами появится перечень доступных шаблонов. Для использования требуемого шаблона выберите его и щелкните на кнопке **Загрузить** (Download). На рис. 4.3 показаны некоторые шаблоны, доступные в категории **Счета** (Invoices).

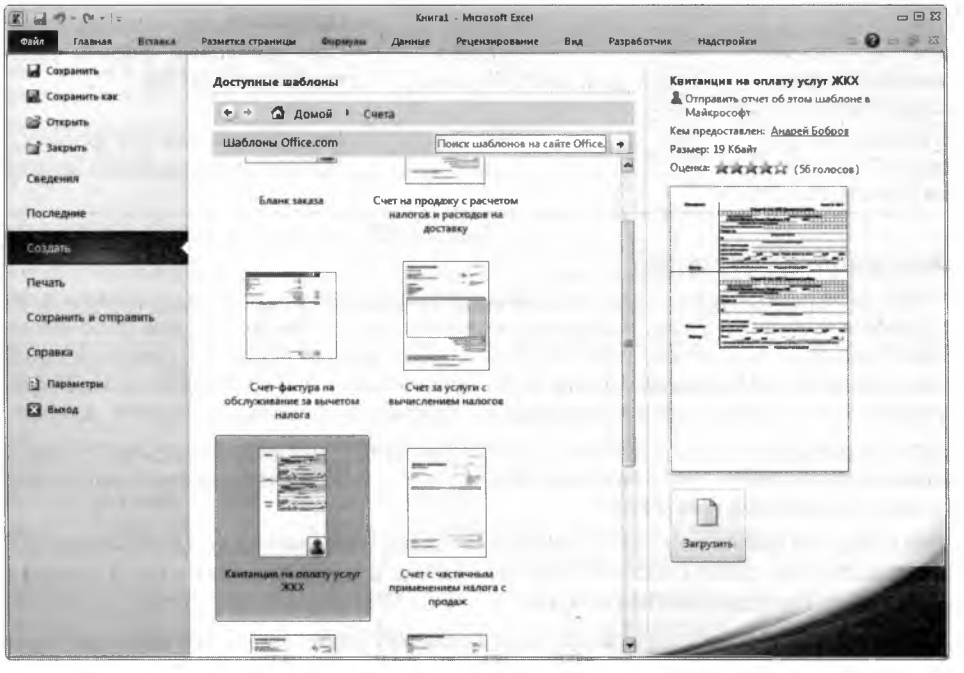


Рис. 4.3. На основе этих шаблонов можно создавать различные счета

Категория Microsoft Office Online включает великое множество шаблонов, однако не все из них эффективны. Поэтому, если вам до сих пор попадались шаблоны не лучшего

качества, не расстраивайтесь. Даже посредственный шаблон путем несложных манипуляций можно существенно улучшить. Этот путь зачастую намного проще, чем создание рабочей книги "с нуля".

Примечание

Местонахождение папки `Шаблоны` варьируется в зависимости от версии Excel. Для поиска этой папки воспользуйтесь следующим VBA-кодом:

```
MsgBox Application.TemplatesPath
```

Создание шаблонов

В Excel поддерживаются следующие три типа шаблонов.

- **Шаблон рабочей книги, заданный по умолчанию.** Этот шаблон применяется в качестве основы для создания рабочих книг. Соответствующий файл именуется `книга1.xltx`.
- **Шаблон рабочего листа, заданный по умолчанию.** Используется для создания рабочих листов, включаемых в рабочую книгу. Соответствующий файл именуется `лист1.xltx`.
- **Пользовательские шаблоны рабочих книг.** Как правило, в данном случае речь идет о готовых к применению рабочих книгах, которые включают формулы, а их структура определяется пользователем. При использовании подобного шаблона пользователю останется подставить необходимые значения и получить требуемый результат.

Использование шаблона для изменения рабочей книги, заданной по умолчанию

Каждая созданная вами рабочая книга имеет ряд настроек, заданных по умолчанию. Например, она включает три рабочих листа, на которых видны линии сетки, для ввода текста используется шрифт Calibri размером 11 пунктов, ширина столбцов составляет 8,43 см и т.д. Если вам не подходят стандартные значения, можете их изменить.

Изменение стандартных настроек в рабочей книге Excel, заданной по умолчанию, не представляет особого труда, а также позволяет сэкономить массу времени. Ниже описаны выполняемые в этом случае действия.

1. Откройте новую рабочую книгу.
2. Добавьте или удалите рабочие листы, после чего их количество будет равно желаемому.
3. Выполните другие изменения, включая настройку ширины столбцов, именованных стилей, параметров страницы, а также многих других параметров, доступных в разделе **Дополнительно** (Advanced) диалогового окна **Параметры Excel** (Excel Options).

   Для изменения стандартного форматирования, примененного к ячейке, выберите команду **Главная⇨Стили⇨Стили ячеек** (Home⇨Styles⇨Cell Styles), после чего измените настройки для стиля `Обычный` (Normal). Например, можно изменить шрифт, заданный по умолчанию, его размер либо числовой формат.
4. Как только рабочая книга приобретет нужный вид, выберите команду **Файл⇨Сохранить как** (Office⇨Save As).

5. В диалоговом окне Сохранение документа (Save As) в поле Тип файла (Save As Type) выберите запись Шаблон Excel (Template (*.xltx)).
6. Введите имя файла `книга.xltx`.
7. Сохраните файл в папке `\XLStart` (не в папке `Шаблоны`).
8. Закройте файл.

Совет

Для определения точного местонахождения папки `\XLStart` воспользуйтесь следующим VBA-кодом:

```
MsgBox Application.StartupPath
```

После выполнения всех описанных выше действий новая рабочая книга предстанет пред ваши светлы очи после запуска Excel на базе шаблона рабочей книги `книга.xltx`. Для создания рабочей книги на основе подготовленного шаблона можно также нажать клавиши <Ctrl+N>. Если же возникнет необходимость вернуться к стандартной рабочей книге, заданной по умолчанию, просто удалите файл `книга.xltx`.

Примечание

Если после выбора команды Файл⇨Создать (File⇨New) в диалоговом окне Создание книги (New Workbook) выбрать пункт Новая книга (Blank Workbook), будет создана рабочая книга не на основе шаблона `книга.xltx`.

Использование шаблона для изменения рабочего листа, заданного по умолчанию

После вставки нового рабочего листа в рабочую книгу программа Excel использует настройки рабочего листа, заданные по умолчанию. Здесь имеются в виду такие параметры, как ширина столбца, высота строки и т.д. Для изменения стандартных настроек выполните следующие действия.

1. Создайте рабочую книгу, в которой удалите все рабочие листы, за исключением одного.
2. Выполните все изменения, включая настройку ширины столбцов, именованных стилей, параметров страницы, а также настроек в диалоговом окне Параметры Excel (Excel Options).
3. После завершения настройки рабочей книги выберите команду Файл⇨Сохранить как (File⇨Save As).
4. В диалоговом окне Сохранение документа (Save As) в поле Тип файла (Save As Type) выберите запись Шаблон Excel (Template (*.xltx)).
5. Введите имя файла **`лист.xltx`**.
6. Сохраните файл в папке `\XLStart` (не в папке `Шаблоны`).
7. Закройте файл.
8. Завершите и снова запустите Excel.

По окончании этой операции все новые листы, которые включаются в рабочую книгу после щелчка на кнопке Вставить лист (Insert Worksheet) (эта кнопка находится правее от ярлыка последнего листа), будут отформатированы в соответствии с шаблоном

лист.xltx. Для вставки нового рабочего листа можно также нажать клавиши <Shift+F11>.

Создание шаблонов рабочих книг

Шаблоны книга.xltx и лист.xltx, рассмотренные в предыдущих разделах, представляют собой специальные типы шаблонов, которые определяют настройки, заданные по умолчанию, для новых рабочих книг и рабочих листов. Теперь же поговорим о других типах шаблонов, называемых *шаблонами рабочих книг* и представляющих собой обычные рабочие книги, на основе которых создаются новые рабочие книги либо рабочие листы.

Для чего нужен шаблон рабочей книги? Главным образом для того, чтобы избавить вас от рутинной работы. Предположим, требуется создать ежемесячный отчет о продажах, который содержит данные о продажах вашей компании по регионам, а также результаты вычислений и диаграммы. В этом случае можно создать шаблон, который выполняет все эти операции (за исключением ввода начальных данных). Как только потребуется создать очередной отчет, откройте рабочую книгу, основанную на созданном шаблоне, заполните пустые поля и можете отдыхать.

Примечание

Можно воспользоваться предыдущим ежемесячным отчетом о продажах, сохранив его под новым именем. При этом велика вероятность ошибки, поскольку можно просто забыть о команде Сохранить как (Save As) и случайно перезаписать файл, содержащий данные ежемесячного отчета за прошлый месяц. Еще одна возможность заключается в использовании команды Из существующего документа (New From Existing), значок которой находится в окне Создание документа (New Workbook). При этом создается рабочая книга на основе существующей, причем этой книге автоматически дается новое имя, исключающее возможность случайной перезаписи.

При создании рабочей книги на основе шаблона ее имя (заданное по умолчанию) образуется на основе имени шаблона, к которому добавляется число. Например, при создании рабочей книги на основе шаблона Отчет о продажах.xltx заданным по умолчанию именем для рабочей книги будет Отчет о продажах1.xlsx. При первом сохранении рабочей книги, созданной на основе шаблона, отображается диалоговое окно Сохранение документа (Save As). В это время можно переименовать шаблон, если вы того желаете.

Пользовательский шаблон — это, по сути, обычная рабочая книга, поэтому можно воспользоваться всеми возможностями, предоставляемыми Excel (диаграммами, формулами и макросами). Обычно шаблон устанавливается таким образом, что пользователь может вводить значения и тут же получать результаты. Другими словами, большинство шаблонов включают все необходимое, кроме данных, вводимых пользователем.

Примечание

Если шаблон содержит макросы, его следует сохранить в формате Шаблон Excel с поддержкой макросов (Excel Macro-Enabled Template) с расширением XLTM.

Содержимое файла Excel

В Excel 2010 используется XML-формат, на основе которого создаются рабочие книги, шаблоны и надстройки. Фактически эти файлы представляют собой ZIP-архивы. При необходимости они могут быть разархивированы и просмотрены.

В версиях Excel, предшествующих Excel 2007, применялся двоичный файловый формат. И хотя спецификации этого формата известны, работать с двоичными файлами совсем непросто. С другой стороны, файловый XML-формат относится к категории так называемых *открытых форматов*. Подобные файлы могут создаваться и обрабатываться с помощью любых программ, не относящихся к Office 2010.

Структура файла

В настоящем разделе описаны компоненты типичного файла рабочей книги с поддержкой макросов Excel (XLSM). Соответствующая рабочая книга, именуемая `sample.xlsm`, показана на рис. 4.4. Она состоит из одного рабочего листа, одного листа диаграммы и простого макроса на языке VBA. Рабочий лист включает таблицу, кнопку (из группы элементов управления Формы (Forms)), рисунок SmartArt, а также фотографию цветка.

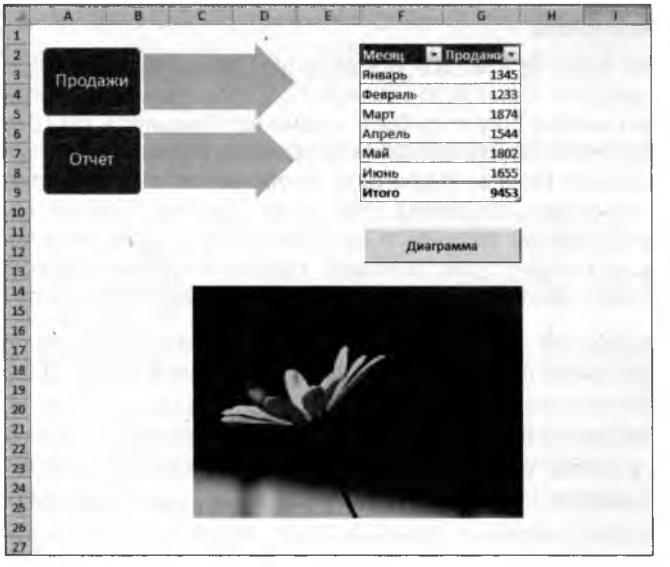


Рис. 4.4. Перед вами пример простой рабочей книги

Компакт-диск

Рабочая книга `sample.xlsm` находится на прилагаемом компакт-диске.

Для просмотра "внутренностей" файла Excel 2010 откройте окно Проводника Windows и добавьте расширение `ZIP` в список имен файлов. После этого файл `sample.xlsm` будет переименован в `sample.xlsm.zip`. Затем можно открыть этот файл с помощью любой программы, выполняющей разархивирование. Автор использовал свойство разархивирования, встроенное в Windows 7.

Примечание

Если в вашей системе выбран режим сокрытия расширений файлов, отключите его. В окне Проводника выберите пункты меню Сервис⇨Параметры папок (Tools⇨Folder Options) и перейдите на вкладку Вид (View). В разделе Файлы и папки (File and Folders) отмените установку флажка Скрывать расширения для зарегистрированных типов файлов (Hide Extensions For Known File Types).

Совет

Возможно, вы захотите разархивировать сжатые файлы, поскольку при этом облегчается их просмотр. При работе в Windows для этого достаточно щелкнуть правой кнопкой на имени файла и выбрать пункт Извлечь файлы (Extract Files).

Первое, что бросается в глаза при просмотре файла, — это наличие структуры каталогов. На левой панели на рис. 4.5 показана полностью развернутая структура каталогов файла рабочей книги. Причем она может изменяться в зависимости от просматриваемой рабочей книги.

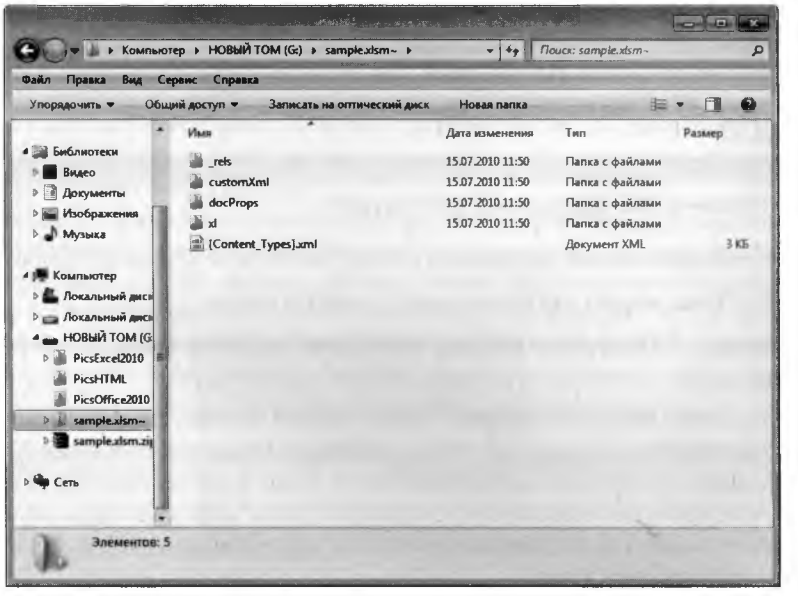


Рис. 4.5. Структура каталогов файла рабочей книги

За небольшими исключениями все используемые файлы являются текстовыми XML-файлами. Их можно просматривать в окне текстового редактора, XML-редактора, веб-браузера либо даже в окне Excel. На рис. 4.6 показано содержимое такого файла, просматриваемого в окне браузера Firefox. Файлы формата, отличного от XML, включают графические файлы и VBA-проекты (хранятся в двоичном формате).

Рассматриваемый XML-файл включает три корневые папки; некоторые из них включают подпапки. Обратите внимание, что многие папки включают папку `_rels`. Здесь находятся XML-файлы, которые определяют связи с другими компонентами пакета.

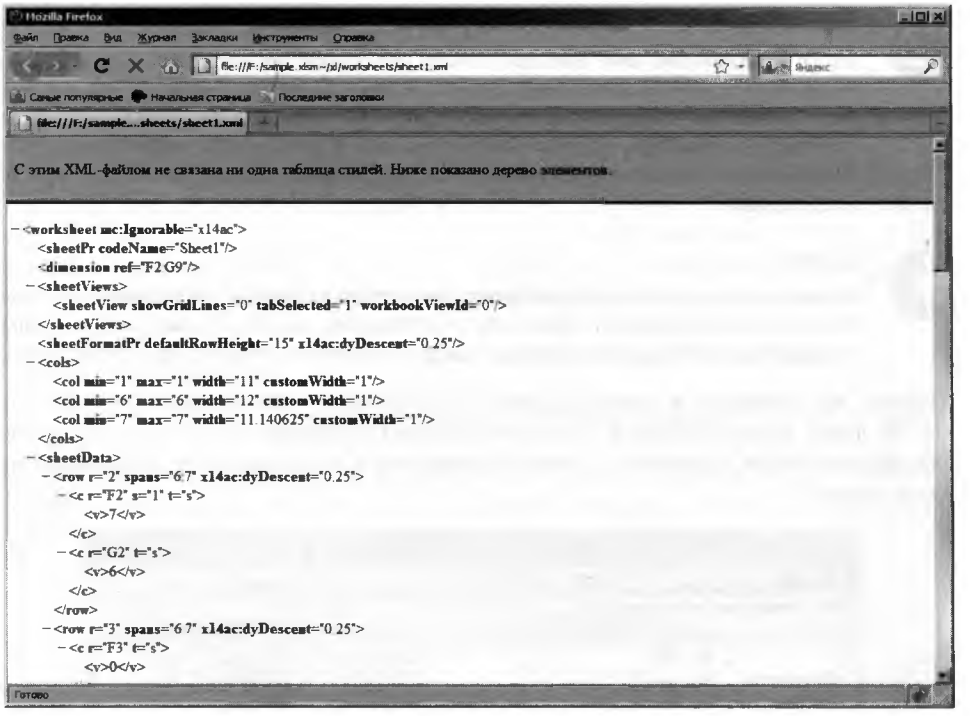


Рис. 4.6. Просмотр XML-файла в окне веб-браузера

Ниже перечислены папки, входящие в состав файла рабочей книги `sample.xlsm`.

- `_rels`. Здесь можно найти сведения о связях в пакете.
- `docProps`. XML-файлы, которые описывают свойства файла и настройки приложения.
- `xl`. Эта папка является "сердцем" файла. Ее имя изменяется в зависимости от типа документа Office (`xl`, `ppt`, `word` и т.д.). Здесь находится несколько XML-файлов, содержащих настройки для рабочей книги. Если в состав рабочей книги включен VBA-код, он будет находиться в двоичном файле с расширением BIN. Эта папка включает несколько подпапок (количество подпапок изменяется в зависимости от выбранной рабочей книги).
  - `charts`. Здесь находится XML-файл для каждой диаграммы, включающий описание ее настроек.
  - `chartsheets`. Здесь содержится XML-файл, включающий данные для каждого листа диаграммы в рабочей книге.
  - `diagrams`. Здесь находятся XML-файлы, в которых содержится описание диаграмм (рисунков SmartArt) в рабочей книге.
  - `drawings`. Здесь содержится XML-файл, включающий данные для каждого "рисунка". Здесь под этим термином подразумеваются кнопки, диаграммы и изображения.
  - `media`. Здесь содержатся внедренные медиаклипы, например GIF- и JPG-файлы.

- `tables`. Включает XML-файл, содержащий данные для каждой таблицы.
- `theme`. Содержит XML-файл, включающий данные о теме рабочей книги.
- `worksheets`. Включает XML-файл для каждого рабочего листа в книге.

Совет

Если добавить расширение `ZIP` к файлу Excel, его, как и раньше, можно будет открыть в Excel, поскольку работа этой программы не зависит от расширения открываемого файла. Также можно сохранить рабочую книгу с расширением `ZIP`. Для этого в диалоговом окне Сохранение документа (Save As) добавьте расширение `ZIP` и заключите имя файла в двойные кавычки, например "МояРабочаяКнига.xlsx.zip".

Почему файловый формат столь важен

"Открытые" файловые XML-форматы, появившиеся в Microsoft Office 2007, представляют собой огромный шаг вперед, важный для всего компьютерного сообщества. Самое главное — рабочие книги Excel в этих форматах относительно легко считывать и записывать посредством ряда программ, отличных от Excel. Например, вполне возможно написать такую программу, которая будет изменять тысячи рабочих книг Excel, причем сама программа Excel не потребуется. Подобная программа может вставлять новые рабочие листы в каждый такой файл. Конечно, разработчик такой программы должен в совершенстве знать структуру XML-файла, но даже если он не знаком с этими вопросами, изучить их не составит особого труда.

Важно также то, что новые форматы файлов более устойчивы к возможным повреждениям (по сравнению с устаревшими двоичными форматами). Я сохранил файл рабочей книги и удалил один из XML-файлов рабочего листа. При попытке повторного открытия этого файла в Excel отобразилось сообщение, показанное на рис. 4.7. В нем говорится о том, что файл был поврежден и данные в нем отличаются от данных в файлах с расширением `.res`. Причем Excel способна "отремонтировать" и открыть файл. При этом удаленный рабочий лист будет помещен на место, хотя и окажется пустым.

Рис. 4.7. Программа Excel может сама "отремонтировать" поврежденный файл рабочей книги

Кроме того, размер заархивированного XML-файла зачастую меньше размера соответствующего двоичного файла. И еще одно преимущество заключается в том, что структурированная природа файла позволяет извлекать отдельные его элементы (например, графику).

Как правило, у пользователя Excel не возникает потребности в просмотре либо изменении XML-компонентов файла рабочей книги. Но если вы являетесь разработчиком, то вам придется создавать код, который изменяет компоненты пользовательского "ленточного" интерфейса Excel. При этом следует хотя бы поверхностно знать структуру XML-файла рабочей книги.

Перекрестная ссылка

Обратитесь к главе 22 за дополнительными сведениями о способах изменения ленты.

Файл OfficeUI

В файле `Excel.officeUI` сохраняются результаты изменений, внесенных на панель быстрого доступа и ленту. Этот XML-файл можно найти в следующей папке:

`C:\Users\<имя_пользователя>\AppData\Local\Microsoft\Office`

Этот файл изменяется после внесения изменений на панель быстрого доступа или ленту. Причем изменения вносятся в файл незамедлительно, еще до закрытия Excel. Обратите внимание, что этот файл появляется только после того, как вы внесете как минимум одно изменение в интерфейс пользователя.

Для просмотра файла `Excel.officeUI` можно воспользоваться редактором XML-кода, веб-браузером или Excel. Просто выполните следующие действия.

1. Создайте копию файла `Excel.officeUI`.
2. Добавьте расширение XML к копии файла, в результате чего название файла примет вид `Excel.officeUI.xml`.
3. Выполните команду **Файл⇨Открыть** (File⇨Open) для открытия файла либо просто перетащите его в окно Excel.
4. Отобразится диалоговое окно, включающее ряд параметров; выберите **XML-таблица** (As an XML Table).

На рис. 4.8 показан импортированный файл `Excel.officeUI` (отображается в виде таблицы). В рассматриваемом примере на панель быстрого доступа добавлено пять команд (строки 4–6 и 12–13 таблицы), а также две вкладки и семь групп (строки 15–23 таблицы).

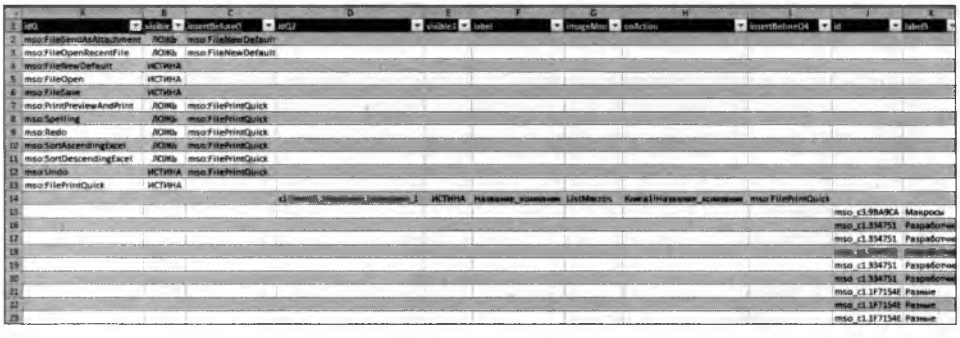


Рис. 4.8. Просмотр файла данных `Excel.officeUI` в Excel

Один и тот же файл `Excel.OfficeUI` может применяться несколькими пользователями одновременно. Например, панель быстрого доступа нетрудно снабдить двумя-тремя десятками полезных инструментов, а ленту — несколькими полезными вкладками, включающими ряд пользовательских групп. Если подобная обновленная панель произведет впечатление на ваших коллег, просто передайте им копию файла `Excel.OfficeUI` и расскажите, куда ее нужно скопировать. Учтите, что в случае копирования

переданной вами копии поверх существующего файла Excel.OfficeUI все изменения, ранее внесенные в интерфейс пользователя, будут утеряны.

Не пытайтесь изменить файл Excel.OfficeUI, если вы точно не представляете, для чего он предназначен. Но при этом не бойтесь экспериментировать. Если при запуске Excel появится сообщение об ошибке в файле Excel.officeUI, можете просто удалить его, после чего Excel создаст новый экземпляр этого файла. Но лучше все же хранить копию исходного файла в безопасном месте.

Файл XLB

Программа Excel хранит настройки панелей инструментов и меню в файле с расширением XLB. Даже несмотря на то, что Excel 2010 официально не поддерживает панели инструментов и меню так, как в предыдущих версиях, файл XLB по-прежнему используется. Если вы не можете его найти, значит, программа до сих пор не сохранила ни одно из пользовательских меню или панелей инструментов.

В момент закрытия Excel текущая конфигурация панелей инструментов сохраняется в файле Excel14.xlb. Этот файл (обычно) находится в следующем каталоге:

C:\Users*<имя пользователя>*\AppData\Roaming\Microsoft\Excel

Этот двоичный файл содержит сведения о положении и видимости всех пользовательских панелей инструментов и меню, а также изменения, которые были добавлены во встроенные панели инструментов и меню.

Файлы надстроек

Надстройка фактически является рабочей книгой Excel, имеющей некоторые особенности.

- Значение свойства рабочей книги IsAddin равно Истина. Это означает, что надстройка может быть загружена и выгружена с помощью диалогового окна Надстройки (Add-Ins).
- Эта рабочая книга скрыта, причем подобное состояние не может изменяться пользователем. Следовательно, надстройка никогда не может быть активной рабочей книгой.
- Если вы работаете с VBA, имейте в виду, что надстройка не входит в коллекцию Workbooks.

Совет

Для получения доступа к диалоговому окну Надстройки (Add-Ins) выберите команду Файл⇨Параметры Excel (File⇨Excel Options). Выберите раздел Надстройки (Add-Ins), в списке Управление (Manage) выберите пункт Надстройки Excel (Excel Add-Ins) и щелкните на кнопке Перейти (Go). Если в рабочем окне Excel отображается вкладка Разработчик (Developer), воспользуйтесь командой Разработчик⇨Надстройки⇨Надстройки (Developer⇨Add-Ins⇨Addins) либо, что проще всего, удобной комбинацией клавиш <Alt+TI>, которая осталась со времен Excel 2003.

Настройки Excel в системном реестре

В диалоговом окне Параметры Excel (Excel Options) находятся десятки настроек, определенных пользователем. Для хранения этих настроек и обращения к ним во время запуска Excel используется реестр Windows. В следующем разделе приводятся некоторые сведения о реестре Windows, а также рассматривается, каким образом Excel использует реестр для хранения настроек.

Кратко о системном реестре

Реестр Windows, который еще называется системным реестром, фактически представляет собой централизованную иерархическую базу данных, используемую операционной системой и приложениями. Реестр появился еще в Windows 95, в которой заменил прежние INI-файлы с хранящимися в них настройками Windows и приложений.

Перекрестная ссылка

Считывание и запись данных в системный реестр может производиться с помощью VBA-макросов. Соответствующие сведения можно найти в главе 11.

Для просмотра системного реестра можно использовать редактор реестра (Registry Editor). Эта программа может применяться также для редактирования содержимого реестра. Файл этой программы называется `regedit.exe`. Прежде чем начать эксперименты, прочтите врезку "Перед изменением реестра...". На рис. 4.9 показано окно редактора реестра.

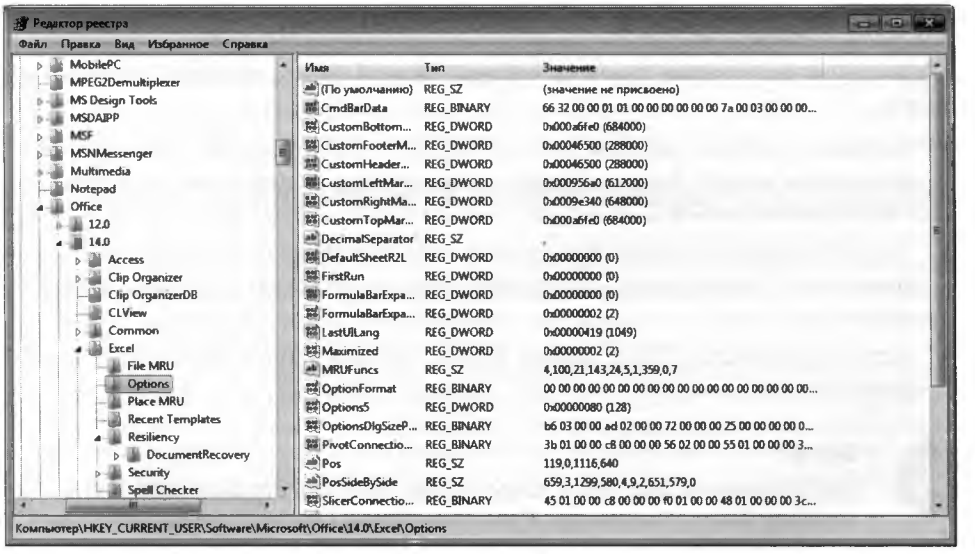


Рис. 4.9. Редактор реестра позволяет просматривать и изменять системный реестр

Перед изменением реестра...

Для выполнения изменений в реестре, включая жизненно важную системную информацию, можно использовать программу `regedit.exe`. Если вы измените не то, что нужно, Windows может работать неправильно.

Используйте команду Файл⇨Экспорт (File⇨Export) редактора реестра. Эта команда позволяет сохранить ASCII-версию всего реестра либо выбранную вами ветвь. Чтобы вернуть реестр в первоначальное состояние, импортируйте ASCII-файл, после чего реестр примет свой исходный вид (команда Файл⇨Импорт (File⇨Import). Дополнительные сведения можно найти в справке редактора реестра.

Системный реестр содержит ключи и значения, расположенные в иерархическом порядке. Ниже приведены ключи верхнего уровня:

- `HKEY_CLASSES_ROOT`;
- `HKEY_CURRENT_USER`;
- `HKEY_LOCAL_MACHINE`;
- `HKEY_USERS`;
- `HKEY_CURRENT_CONFIG`.

Настройки Excel

Информация, используемая Excel 2010, хранится в следующем разделе реестра:

`HKEY_CURRENT_USER\Software\Microsoft\Office\14.0\Excel`

В этом разделе реестра находится ряд ключей, определяющих значения, которые влияют на порядок функционирования Excel.

Настройки реестра обновляются автоматически после закрытия Excel.

Примечание

Учтите, что Excel считывает содержимое реестра Windows один раз — при запуске. Кроме того, Excel единственный раз обновляет настройки реестра — при нормальном завершении. Если Excel завершается аварийно (такое бывает), информация в реестре не обновляется. Если изменить одну из настроек Excel, например отображение строки формул, это изменение не зафиксируется в системном реестре до тех пор, пока Excel не завершит свою работу без эксцессов.

В табл. 4.6 перечислены настройки реестра, имеющие отношение к Excel 2010. Учтите, что некоторых из них вы можете не найти в своей базе данных реестра.

Таблица 4.6. Информация о конфигурации Excel в системном реестре

Раздел	Описание
`Add-in Manager` (Диспетчер надстроек)	Отображает надстройки, которые находятся в диалоговом окне Надстройки (Add-Ins). Надстройки, входящие в комплект поставки Excel, в этом списке отсутствуют. Если нужно удалить какую-либо надстройку, удалите соответствующую запись в окне редактора реестра
`Converters` (Конвертеры)	Здесь перечисляются дополнительные (внешние) конвертеры файлов, которые не встроены в Excel
`Error Checking` (Проверка ошибок)	Настройки, определяющие процесс поиска ошибок в формулах
`File MRU` (Последние открытые файлы)	Сведения о последних использовавшихся файлах (эти файлы отображаются в списке Последние книги (Recent Workbooks), который появляется после выбора команды Файл⇨Последние (File⇨Recent))

Окончание табл. 4.6

Раздел	Описание
`Options` (Параметры)	В этом разделе содержится великое множество параметров, определяющих настройки Excel
`Recent Templates` (Последние шаблоны)	Информация о последних использовавшихся шаблонах
`Resiliency` (Устойчивость)	Информация, применяемая для восстановления документов
`Security` (Безопасность)	Параметры безопасности, определяющие порядок открытия файлов с макросами
`Spell Checker` (Проверка грамматики)	Информация, описывающая параметры модуля проверки грамматики
`StatusBar` (Строка состояния)	Выбранная пользователем информация, которая отображается в строке состояния
`UserInfo` (Информация о пользователе)	Информация о пользователе

Хотя большинство настроек может изменяться в диалоговом окне **Параметры Excel** (Excel Options), некоторые настройки невозможно изменить таким образом (в этом случае применяется редактор реестра). Например, при выделении диапазона ячеек иногда требуется, чтобы выделенные ячейки окрашивались в черный цвет на белом фоне. Для этого достаточно добавить в системный реестр следующий ключ.

1. Откройте редактор реестра и найдите раздел `HKEY_CURRENT_USER\Software\Microsoft\Office\14.0\Excel\Options`.
2. Щелкните правой кнопкой мыши и выберите пункт **Создать⇨Параметр DWORD** (New⇨DWORD Value).
3. Назовите создаваемое значение `Options6`.
4. Щелкните правой кнопкой мыши на ключе `Options6` и выберите пункт **Изменить** (Modify).
5. В диалоговом окне **Изменение параметра DWORD** (Edit DWORD Value) установите переключатель **Десятичная** (Decimal) и введите значение **16** (рис. 4.10).

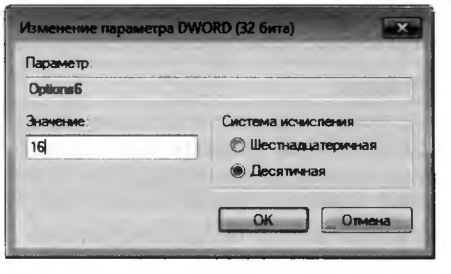


Рис. 4.10. Ввод значения для параметра системного реестра

После перезагрузки Excel ячейки выделяются черным цветом (а не светло-голубым, как ранее). Если вас это не устраивает, удалите запись реестра `Options6`.

Совет

Если появляются проблемы при запуске Excel, причина может быть в повреждении ключей системного реестра. Возможно, придется с помощью редактора реестра удалить следующий раздел реестра:

```
HKEY_CURRENT_USER\Software\Microsoft\Office\14.0\Excel
```

При следующем запуске Excel удаленные ключи реестра будут созданы вновь. При этом существует риск потери некоторой информации.

Часть II

Разработка приложений Excel

В этой части...

Глава 5

Приложения электронных таблиц

В этой главе...

- О приложениях электронных таблиц
- Разработчик и конечный пользователь
- Решение проблем с помощью Excel
- Основные типы электронных таблиц

В этой главе речь пойдет об особенностях использования электронных таблиц на практике.

О приложениях электронных таблиц

Приложение электронных таблиц — это один *файл* или группа связанных *файлов электронных таблиц*, разработанных таких образом, чтобы пользователь, не являющийся их разработчиком, без особой подготовки мог выполнить необходимые действия. Согласно этому определению большинство разработанных вами электронных таблиц, вероятно, такими приложениями не являются. На жестком диске у вас могут быть сохранены десятки или сотни файлов электронных таблиц, но наверняка большинство из них не предназначены для других пользователей.

Ниже приведены характеристики эффективного приложения электронных таблиц.

- Позволяет конечному пользователю выполнить задание, которое, вероятно, он не смог бы выполнить по-другому.
- Предоставляет эффективное решение проблемы (среда электронных таблиц не всегда является оптимальной структурой управления данными).
- Выполняет только ожидаемые действия. Такое требование, возможно, покажется очевидным, но очень часто именно по этой причине приложения нельзя назвать эффективными.

- Выдает точные результаты и не имеет программных ошибок.
- Для выполнения своей работы использует четкие и эффективные методы и алгоритмы.
- Выявляет ошибки, вызванные своим присутствием в системе, не дожидаясь вмешательства пользователя.

Примечание

Обратите внимание, что ошибки и программные ошибки — это не одно и то же. Попытка деления на нуль — *ошибка*, связанная с работой приложения. А то, что такая попытка вовремя не пресечена, является уже *программной ошибкой*.

- Не разрешает пользователю случайно (или умышленно) удалять или видоизменять важные компоненты.
- Имеет простой и понятный графический интерфейс, поэтому пользователь всегда знает, что делать дальше.
- Формулы, макросы и элементы пользовательского интерфейса хорошо документированы, что предоставляет возможность изменять их в случае необходимости.
- Приложение разработано с учетом того, что его можно просто модифицировать, не прибегая к крупномасштабным изменениям (ведь пользователю рано или поздно потребуется внести изменения).
- Располагает легкодоступной справочной системой, которая предоставляет полезную информацию по основным процедурам.
- Должно быть переносимым и работать в любой компьютерной системе, в которой установлены все необходимые программы (в данном случае — требуемая версия Excel).

Не следует удивляться тому, что приложения электронных таблиц можно создавать для различного применения (и в шаблонах, заполняемых данными, и в довольно сложных приложениях с пользовательскими меню и диалоговыми окнами, причем сами приложения могут даже и не выглядеть, как электронные таблицы).

Разработчик и конечный пользователь

Мы часто употребляем термины *разработчик* и *конечный пользователь*. Читателей данной книги можно смело зачислить в отряд разработчиков или как минимум потенциальных разработчиков.

Итак, определимся с терминологией. Разработчик — это специалист, который создает приложение электронных таблиц. В совместных проектах число разработчиков обычно больше одного (команда разработчиков). Конечный пользователь (его для краткости будем называть просто пользователем) — это человек, применяющий результаты деятельности разработчика по программированию электронных таблиц. Во многих случаях конечных пользователей бывает достаточно много, а разработчиком часто является один из пользователей.

Кто такие разработчики и чем они занимаются

Вот уже более двадцати лет я преподаю методологию разработки. Поэтому чаще всего я имею дело с теми, кто называет себя разработчиками электронных таблиц. Среди них различаются две основные группы.

- **Штатные специалисты**, которые тесно сотрудничают с пользователями и основательно знают их потребности. Во многих случаях эти разработчики также являются пользователями своего приложения. Часто они разрабатывают приложение, чтобы решить только одну конкретную проблему.
- **Специалисты со стороны**, как правило, приглашаются с целью решения проблемы. В большинстве случаев такие разработчики знакомы с вопросом лишь в общих чертах, однако хорошо знают специфику приложения, которое разрабатывают. Разработчики могут быть сотрудниками той же самой компании, которой требуется приложение, но только другого подразделения.

У одних специалистов на разработку приложения уходит все рабочее время. Они могут быть как внутренними специалистами, так и специалистами со стороны. Достаточно много консультантов (со стороны) неплохо зарабатывают, работая "свободными художниками" по созданию приложений электронных таблиц.

Другие же разработчики электронных таблиц не посвящают этому делу все свое рабочее время и даже не осознают, что они разрабатывают соответствующие приложения. Такими разработчиками часто выступают компьютерные "знатоки", которые все знают о компьютерах и программах. Часто эти люди создают приложения электронных таблиц как раз для того, чтобы облегчить себе жизнь. Ведь время, которое они тратят, чтобы разработать для других сотрудников хорошее приложение, часто экономит часы, которые пришлось бы потратить на обучение специалистов. Кроме того, время разработки такого приложения значительно меньше времени, которое требуется, чтобы отвечать на вопросы сотрудников.

Разработчики электронных таблиц обычно принимают участие в следующих операциях, самостоятельно выполняя большинство из них или даже все:

- определение потребностей пользователя;
- планирование приложения, которое соответствует этим потребностям;
- определение наиболее подходящего интерфейса пользователя;
- создание электронной таблицы, формул, макросов и пользовательского интерфейса;
- тестирование приложения в разных условиях;
- изменение приложения с целью повышения его надежности и отказоустойчивости (часто по результатам тестирования);
- обеспечение эстетической привлекательности и наглядности приложения;
- придание приложению красивого внешнего вида и интуитивности;
- документирование усилий, потраченных на разработку;
- размещение приложения в компьютере пользователя;
- обновление приложения в случае необходимости.

Перекрестная ссылка

Более подробно об этих операциях речь идет в главе 6.

Разработчики должны хорошо знать среду, в которой они работают (в данном случае — программу Excel). Конечно, по уверениям компании Microsoft, использовать эту программу очень легко, но определение легкости отличается для каждого конкретного

пользователя. Для разработки нестандартных приложений электронных таблиц с помощью Excel требуется глубокое знание формул, функций, макросов, пользовательских диалоговых окон, пользовательских панелей инструментов, а также надстроек и команд меню. Большинство пользователей, как правило, не соответствуют выдвигаемым требованиям и не имеют намерения изучать эти подробности. Итак, перейдем к следующей теме — классификации пользователей электронных таблиц.

Классификация пользователей электронных таблиц

Пользователи, работающие с электронными таблицами (как пользователи-разработчики, так и обычные пользователи), различаются по *степени* (или *опытности*) использования электронных таблиц и по *интересу к изучению* самих таблиц.

Каждый из этих критериев имеет три уровня. Комбинируя уровни обоих параметров, получаем девять вариантов, которые представлены в табл. 5.1. Однако рассматривать мы будем только семь из них. Дело в том, что минимальный интерес к электронным таблицам обычно проявляют те пользователи, которые имеют опыт работы с ними (интерес как раз и стимулировал таких пользователей к приобретению опыта). Что же касается пользователей, которые обладают большим опытом работы с электронными таблицами и низким уровнем интереса, то из них обычно получаются не очень хорошие разработчики.

Таблица 5.1. Классификация пользователей электронных таблиц по опытности и интересу к изучению предмета

Уровень опыта	Интерес отсутствует	Умеренный интерес	Очень большой интерес
Небольшой опыт	Пользователь	Пользователь	Пользователь/Потенциальный разработчик
Умеренный опыт	Н/Д	Пользователь	Разработчик
Очень большой опыт	Н/Д	Пользователь	Разработчик

Очевидно, что у разработчиков электронных таблиц должен быть как немалый опыт работы с этими таблицами, так и высокий к ним интерес. Те, у кого небольшой опыт работы, но имеется интерес, являются потенциальными разработчиками. Все, что им необходимо, — приобрести опыт. И если вы читаете эту книгу, то, вероятно, относитесь к одной из категорий последнего столбца этой таблицы.

Для кого предназначены приложения электронных таблиц

Оставшиеся ячейки последней таблицы соответствуют тем конечным пользователям электронных таблиц, которых вы считаете "потребителями" приложений электронных таблиц. Разрабатывая такое приложение, предназначенное для других людей, необходимо знать, какие из этих групп пользователей действительно будут его применять.

Значительная часть пользователей не имеют опыта и интереса. Это люди, которым электронная таблица необходима для работы. Она рассматривается просто как средство достижения конечной цели. Обычно таким пользователям мало известно о компьютерах и программах, и, как правило, им не интересно изучать то, что не относится к их работе. Возможно, компьютеры их немного пугают. Очень часто им даже неизвестна версия

процессора электронных таблиц, которую они используют; кроме того, они не знают всех возможностей программы. Очевидно, что приложения, разработанные для этой группы, должны быть дружественными к пользователям, т.е. простыми, не внушающими страх и по возможности отказоустойчивыми.

С точки зрения разработчика, более интересной группой являются пользователи, которые обладают умеренным опытом работы с электронными таблицами и заинтересованы в том, чтобы знать больше. Эти пользователи имеют понятие о формулах, умеют работать с функциями надстроек и обычно знают о возможностях программного продукта. Они, как правило, ценят труд, который вы вложили в приложение, и на них часто производят впечатление приложенные вами усилия. Более того, эти пользователи часто предлагают прекрасные идеи, которые помогают улучшить ваш продукт. Приложения, разработанные для этой группы, также должны быть дружественными к пользователям (легкими в использовании и отказоустойчивыми), но это еще не все. Они, кроме того, могут быть более сложными и уникальными, чем приложения, которые предназначены для аудитории, состоящей из менее опытных и заинтересованных пользователей.

Решение проблем с помощью Excel

Выше речь шла о таком базовом понятии, как приложение электронных таблиц; вы узнали о некоторых типах конечных пользователей и разработчиков приложений и выяснили назначение процессоров электронных таблиц. Теперь настало время показать, какие задачи решаются с помощью приложений электронных таблиц.

Вы, возможно, уже имеете достаточно хорошее представление о тех задачах, для решения которых применяется Excel. Традиционно такие программы использовались в приложениях, которые по своей природе являются в значительной степени интерактивными. Хороший пример таких приложений — бюджет корпорации. После подготовки соответствующей модели (т.е. создания расчетных формул) управление бюджетом сводится к введению конечных значений и изучению итогов, полученных в результате автоматических вычислений. Часто разработчикам приложения расчета бюджета достаточно распределить фиксированные ресурсы по разным видам деятельности и представить результаты в привлекательном (или как минимум в удобочитаемом) виде. Конечно же, процессор электронных таблиц является для этого идеальным средством.

Впрочем, задачи, аналогичные описанной, составляют лишь небольшой процент того, для чего разрабатываются электронные таблицы. Зачастую пользователи процессоров электронных таблиц (особенно в последние годы) применяют эти программы для решения не тех задач, для которых они предназначались изначально.

Приведем несколько примеров нетрадиционного применения процессора электронных таблиц Excel.

- **Как средство проведения презентаций.** Например, используя исключительно Excel, вы можете с минимальными усилиями создать привлекательное интерактивное слайд-шоу, выводимое на экран монитора.
- **Как инструмент ввода данных.** Программа электронных таблиц часто является самым эффективным средством решения таких задач, как повторный ввод данных. Введенные данные могут экспортироваться в самые разные форматы, применяемые другими программами.

- **Как диспетчер баз данных.** Если вы имеете дело с достаточно простыми структурами данных, намного проще для управления ими воспользоваться Excel, а не Access.
- **Как генератор бланков.** Многим пользователям для создания привлекательных печатных бланков проще обратиться к инструментам форматирования Excel, а не изучать настольную издательскую систему, например PageMaker.
- **Как текстовый процессор.** Функции управления текстом, которые присутствуют во всех процессорах электронных таблиц, предоставляют возможность манипулировать текстом так, как это невозможно даже в текстовом процессоре.
- **Как платформа для простых игр.** Конечно, разработчики программы Excel о таком ее применении и не думали. Однако я загрузил из Интернета (а также написал собственноручно) интересные стратегические игры, в которых применяются инструменты Excel и других процессоров электронных таблиц.

Вы, вероятно, можете пополнить этот список, вспомнив еще многие другие примеры.

Ирония в том, что универсальность процессоров электронных таблиц является палкой о двух концах. С одной стороны, появляется искушение применить такую программу для решения любой возникшей проблемы. А с другой — вы часто выбиваетесь из сил, пытаясь с помощью электронных таблиц справиться с проблемой, для которой легче найти другое решение.

Основные типы электронных таблиц

В этом разделе приведена классификация электронных таблиц, включающая несколько основных типов. Она проиллюстрирует, каким образом электронные таблицы вписываются в общую картину управления данными с помощью компьютера. Конечно, данная классификация является довольно условной. Она создана исключительно на основе моего личного опыта. Более того, ее категории часто пересекаются, однако к ним относится большинство электронных таблиц.

Предложим такие названия для категорий (типов) электронных таблиц:

- электронные таблицы “на скорую руку”;
- электронные таблицы “не для посторонних глаз”;
- однопользовательские приложения;
- приложения-“спагетти”;
- приложения-утилиты;
- надстройки с функциями рабочих листов;
- одноблоковые бюджеты;
- модели “что если”;
- приложения для хранения данных и доступа к ним;
- клиентские приложения для доступа к базам данных;
- приложения “под ключ”.

О каждой из этих категорий рассказывается в следующих разделах.

Электронные таблицы "на скорую руку"

Вероятно, это самый распространенный тип электронных таблиц. Большинство электронных таблиц из этой категории являются небольшими. Они разрабатываются для того, чтобы быстро решить проблему или получить ответ на вопрос. Рассмотрим пример. Вы собираетесь купить новую машину, поэтому необходимо для разных сумм кредита вычислить размер ежемесячной выплаты. Или вы решили создать диаграмму, которая будет отображать объем продаж вашей компании по месяцам. Введите 12 значений, скопируйте диаграмму и вставьте ее в документ, создаваемый в текстовом процессоре.

В обоих случаях на разработку модели уходит несколько минут, и, конечно же, у вас нет времени документировать свои действия. Скорее всего, вы и не думаете создавать какие-либо макросы или пользовательские диалоговые окна. Наверняка вы даже не считаете нужным сохранять эти простые электронные таблицы на диске. Вот почему электронные таблицы этой категории не являются приложениями.

Электронные таблицы "не для посторонних глаз"

Как следует из названия, электронные таблицы, которые попадают в эту категорию, не увидит и не будет использовать никто, кроме вас — их разработчика. В качестве примера можно привести файл с информацией, которая относится к оплате налогов, начисляемых на основе ваших доходов. Вы открываете свой файл, когда к вам по почте приходит чек, или "влезаете" в расходы, которые можно рассматривать как производственные, или же покупаете у торговцев на улице ворованную автомагнитолу и т.д. Другим примером является электронная таблица, в которой вы ведете учет времени, потраченного вашими сотрудниками (отсутствие на работе по болезни, отпуск и т.д.) попусту или попросту прогулянного.

Электронные таблицы данной категории отличаются, например, от созданных "на скорую руку", которые не являются одноразовыми, поэтому их и сохраняют в файлах. Однако на них не стоит тратить много времени — примените простое форматирование (лишь в случае необходимости). В электронных таблицах этого типа также отсутствуют инструменты обнаружения ошибок: вы знаете, каким образом формулы создавались, поэтому вам хорошо известно, как избежать ввода данных, приводящих к ошибочным результатам. При появлении ошибки вы сразу будете знать, чем она вызвана.

Сложность электронных таблиц этой категории со временем возрастает, однако они не являются приложениями.

Однопользовательские приложения

Это могут быть приложения электронных таблиц, используемые только их разработчиком, однако по своей сложности вышедшие далеко за пределы электронных таблиц "не для посторонних глаз". Например, я разработал рабочую книгу, чтобы вести в ней учет зарегистрированных пользователей условно-бесплатных приложений. Книга начинается простой базой данных, расположенной на одном рабочем листе (она предназначается только для просмотра разработчиком). Однако вскоре эта рабочая книга также понадобилась для создания накладных и почтовых этикеток. Потратив однажды около часа на создание макросов, я понял, что превратил эту рабочую книгу из приложения "не для посторонних глаз" в настоящее однопользовательское приложение.

Создавая для себя однопользовательские приложения, вы имеете прекрасную возможность попрактиковаться с инструментами, которыми пользуются разработчики Excel-приложений. Например, вы можете научиться создавать пользовательские диалоговые окна, видоизменять меню, создавать пользовательские панели инструментов, писать VBA-макросы и т.д.

Совет

Работа над сложным проектом — наилучший способ освоить сложные функции Excel (а также любого другого приложения).

Приложения-"спагетти"

Среди электронных таблиц все еще распространены *приложения-"спагетти"*. Само понятие возникло из-за того, что иногда в отдельных частях приложения разобраться довольно трудно; эти части во многом так же спутаны между собой, как спагетти на тарелке. Большинство этих электронных таблиц разрабатывались как специализированные однопользовательские приложения. Но со временем они перешли во владение других пользователей, которые внесли свои изменения. По мере того как требования менялись, а сотрудники приходили и уходили, одни части появлялись, а другие игнорировались. Спустя некоторое время первоначальное назначение рабочей книги забылось. В результате получился файл, используемый довольно часто, однако никто не знает, как же этот файл работает.

Каждый, кто имеет дело с приложениями-"спагетти", понимает, что их необходимо полностью переделывать. Но поскольку мало кто знает, как это сделать, со временем дела обстоят все хуже и хуже. Консультанты по процессорам электронных таблиц зарабатывают немалые деньги, занимаясь распутыванием таких приложений. Как правило, в процессе улучшения приложений-"спагетти" наиболее эффективным является следующее решение: заново определить, что нужно пользователям, и создать приложение "с нуля".

Приложения-утилиты

Никто и никогда еще не был полностью доволен используемым процессором электронных таблиц. И какой бы хорошей ни была программа Excel, в ней все равно находят недостатки. Поэтому перейдем к следующей категории электронных таблиц — *утилитам*. Это специальные инструменты, которые предназначены для выполнения единственной повторяющейся задачи. Например, если вы часто импортируете текст в Excel, то вам, возможно, требуются специальные команды для обработки текста, в частности для преобразования (без использования формул) выделенного текста в верхний регистр. В данном случае рекомендуется разработать утилиту для обработки текста, которая будет выполнять необходимые задачи.

Примечание

Power Utility Pak — это коллекция приложений/утилит для программы Excel. Они разработаны для того, чтобы расширить функциональность программы. Эти утилиты работают, как обычные команды Excel. Пробную версию пакета Power Utility Pak можно загрузить с веб-сайта автора книги (`www.spreadsheetpage.com`). Желающие могут также получить полный исходный код за небольшую плату.

По своей природе приложения-утилиты являются достаточно универсальными. Что касается макросов, то многие из них предназначены для того, чтобы выполнять конкретную операцию с данными конкретного типа, расположенными в рабочей книге, опять же, конкретного типа. Эффективное приложение-утилита работает так, как обычная команда Excel. Другими словами, утилите необходимо распознать контекст, в котором должна выполняться команда, и выполнить соответствующее действие. Чтобы утилита была в состоянии обрабатывать любые возможные ситуации, обычно используется громоздкий код, предназначенный для обработки ошибок.

В приложениях-утилитах всегда используются макросы; в них также могут применяться пользовательские диалоговые окна. Создать такие утилиты с помощью Excel довольно легко: их следует преобразовать в надстройки и присоединить к пользовательскому интерфейсу Excel, чтобы эти утилиты выглядели, как часть программы.

Перекрестная ссылка

Вопросы разработки утилит очень важны, поэтому их рассмотрению посвящена целая глава. В главе 16 рассматриваются методы создания пользовательских утилит Excel средствами VBA.

Надстройки с функциями рабочих листов

Программа Excel располагает огромным количеством функций рабочих листов, которые можно использовать в формулах. Однако в некоторых случаях необходима определенная функция, а ее поиски дают отрицательный результат. Для этого создайте собственную функцию, используя VBA. Благодаря пользовательским функциям рабочих листов формулы часто становятся проще, а поддерживать электронные таблицы оказывается легче.

Перекрестная ссылка

В главе 10 описывается процесс создания пользовательских функций рабочего листа, а также приводится множество примеров.

Одноблоковые бюджеты

Под *одноблоковым бюджетом* подразумевается рабочий лист (не обязательно модель бюджета), который состоит из одного блока ячеек. Верхняя его строка может быть составлена из имен, относящихся к промежуткам времени (месяцам, кварталам или годам), а левый столбец обычно состоит из категорий определенного типа. Как правило, нижняя строка и правый столбец составлены из итоговых формул. В блоке ячеек могут использоваться формулы подсчета промежуточных итогов.

Данный тип электронных таблиц очень распространен. С учетом как раз этой модели была разработана программа VisiCalc (первый процессор электронных таблиц). В большинстве случаев простые модели одноблоковых бюджетов не являются удачными кандидатами в приложения, так как они слишком просты. Впрочем, существуют исключения. Например, можно было бы преобразовать такую электронную таблицу в приложение, если бы моделью одноблокового бюджета являлась громоздкая трехмерная электронная таблица, в которую необходимо включить сводные данные из других файлов или которой будут пользоваться руководители отделов, возможно, не разбирающиеся в электронных таблицах.

Модели "что если"

Многие считают модель "что если" воплощением всего самого лучшего, что имеется в электронных таблицах. Способность мгновенно пересчитывать тысячи формул делает процессоры электронных таблиц идеальным инструментом для финансового моделирования, а также для других моделей, которые зависят от значений нескольких переменных. Если подумать, то почти каждая электронная таблица с формулами является моделью "что если" (она часто распространяется в виде шаблона). Изменение значения в ячейке, используемой в формуле, имитирует ситуацию, приводящую к возникновению вопроса "Что будет, если?.." Кроме того, приложения данной категории довольно сложные. Они состоят из электронных таблиц, которые специально разрабатывались для прогнозирования влияния отдельных значений на конечный результат.

Модели "что если" являются хорошими кандидатами в приложения, ориентированные на пользователя, особенно если модель будет использоваться продолжительное время. При условии создания для приложения удачного графического интерфейса его сможет легко использовать даже тот, кто совсем не разбирается в компьютерах. Например, можно создать интерфейс, который предоставляет пользователю возможность задавать имена для различных наборов условий, мгновенно просматривать результаты выбранного сценария и одним щелчком на кнопке создавать правильно отформатированные сводные диаграммы.

Электронные таблицы для хранения данных и доступа к ним

Многие рабочие книги Excel состоят из одной или нескольких таблиц баз данных (иногда эти таблицы называются *списками*). Они могут использоваться для обработки любых данных, причем зачастую более простым способом, чем в случае использования СУБД. Если таблицы настроены правильно, для суммирования содержащихся в них данных могут применяться сводные таблицы.

Электронные таблицы этой категории часто являются кандидатами в приложения, особенно если конечным пользователям необходимо выполнять операции умеренной сложности.

Что же касается сложных приложений баз данных, в частности таких, в которых используется огромное количество таблиц с заданными связями между ними, то для них больше подходит настоящий процессор баз данных, например Access.

Клиентские приложения баз данных

Электронные таблицы все чаще применяются для доступа к внешним базам данных. Пользователи электронных таблиц с помощью инструментов Excel получают доступ к данным, хранящимся во внешних файлах, даже если у этих данных разные форматы. Приложение, выполняющее подобные задачи, часто называют *управленческой информационной системой* (Executive Information System — EIS). Такая система комбинирует данные из нескольких источников, генерируя выборки для пользователей.

Доступ из электронной таблицы к внешним базам данных часто вызывает страх у начинающих пользователей. Создание управленческой информационной системы является идеальным способом применения Excel, так как основная цель подобных систем обычно состоит в том, чтобы обеспечить простоту в использовании.

Приложения "под ключ"

Последняя категория электронных таблиц является самой сложной. "Под ключ" подразумевает такую готовность к применению, когда конечному пользователю достаточно минимальной подготовки. Например, при загрузке файла появляется окно, позволяющие сделать совершенно однозначный выбор. Приложения "под ключ" могут выглядеть так, будто они созданы не с помощью процессора электронных таблиц, и часто пользователь взаимодействует не с ячейками, а с диалоговыми окнами.

Электронные таблицы многих описанных выше категорий вполне можно сделать приложениями "под ключ". Среди общих элементов таких приложений самыми главными являются хорошее планирование, обработка ошибок и система пользовательского интерфейса. О них речь пойдет в следующих главах.

Глава 6

Принципы разработки приложений электронных таблиц

В этой главе...

- Этапы разработки приложения
- Определение потребностей пользователя
- Проектирование приложения с учетом потребностей пользователя
- Определение удобного пользовательского интерфейса
- Работа с конечным пользователем
- Другие вопросы разработки приложений

В данной главе приведены общие правила, которые вы, возможно, сочтете полезными, когда будете учиться создавать эффективные приложения, работающие в Excel.

Этапы разработки приложения

К сожалению, не существует простого и безошибочного метода, который гарантировал бы создание эффективного приложения электронных таблиц. У каждого разработчика есть собственный стиль создания таких приложений, и наилучший способ определяет для себя сам разработчик. Кроме того, каждый проект, за который вы беретесь, отличается от других, и поэтому для его реализации требуется особый подход. И наконец, требования и общие представления людей, с которыми (или на которых) вам предстоит работать, также играют определенную роль в процессе разработки.

Как уже отмечалось в предыдущей главе, разработчики электронных таблиц несут ответственность за соблюдение следующих требований:

- определение потребностей пользователя;
- планирование приложения, которое соответствует этим условиям;

- разработка наиболее подходящего интерфейса пользователя;
- создание электронной таблицы, формул, макросов и пользовательского интерфейса;
- тестирование и отладка приложения;
- изменение приложения с целью повышения его надежности и отказоустойчивости (часто по результатам тестирования);
- эстетическая привлекательность и интуитивная ясность приложения;
- документирование усилий, затраченных на разработку;
- разработка пользовательской документации и справочной системы;
- размещение приложения в компьютере пользователя;
- обновление приложения в случае необходимости.

Все эти требования необязательно соблюдать при создании каждого приложения, да и порядок их выполнения в разных проектах может быть разным. Перечисленные действия описаны в следующих разделах. Что же касается технических деталей, то в большинстве случаев их также можно найти в следующих главах.

Определение потребностей пользователя

При разработке проекта приложения электронных таблиц одним из первых действий является точное определение потребностей конечного пользователя. И если вы не сможете заранее оценить потребности аудитории, то позднее это обернется дополнительной работой по устранению недостатков. Именно поэтому оценить потребности необходимо в первую очередь.

В отдельных случаях вы можете быть близко знакомы с конечными пользователями и даже сами можете выступать одним из них. Что же касается остальных случаев (например, вы работаете консультантом, который разрабатывает проект для нового клиента), то о пользователях или об их запросах вы, возможно, вообще ничего не будете знать.

Каким образом можно определить потребности пользователей? Если вам необходимо разработать приложение электронной таблицы, встретьтесь с потенциальными пользователями и опросите их. А еще лучше — изложите пользователям свое видение разрабатываемого приложения в виде печатных документов, создайте блок-схему, уделите внимание мелким деталям и сделайте все, чтобы создать именно такое приложение, какое нужно пользователю.

Ниже приведены основные правила, придерживаясь которых, вы облегчите начальную фазу разработки.

- Не убеждайте себя в том, что уже знаете потребности пользователей. Если на этом этапе основываться на предположениях, в дальнейшем неизбежно возникнуть проблемы.
- Если существует такая возможность, говорите непосредственно с потенциальными клиентами приложения, а не только с руководителем проекта или управляющим фирмы.
- Узнайте, что уже сделано (если только сделано) для удовлетворения потребностей пользователей. Вы, возможно, сэкономите часть своего времени, переделав существующее приложение. В крайнем случае, изучая имеющиеся решения, вы более подробно изучите работу конечных пользователей.

- Определите, какие ресурсы имеются в распоряжении пользователя. Постарайтесь, например, узнать, существуют ли какие-либо аппаратные или программные ограничения, которые нужно учитывать.
- По возможности, определите все типы систем, в которых будет запускаться ваше приложение. В случае, если приложение будет выполняться в низкопроизводительных системах, примите этот факт к сведению. Для получения дополнительных сведений по этой теме обратитесь к разделу "Быстродействие системы".
- Узнайте, какая версия (или версии) Excel используется в системе. Компания Microsoft делает все возможное, чтобы убедить пользователей применять последние версии выпускаемых ею программ. Однако по результатам проведенного опроса последние версии пакета Microsoft Office доступны менее чем половине всех пользователей этого пакета.
- Узнайте уровень квалификации конечных пользователей. Эта информация поможет вам правильно разработать приложение.
- Определите, как долго будет использоваться приложение и нужно ли будет его изменять. Это повлияет на выполняемые операции и поможет спланировать изменения.

И последнее замечание. Не удивляйтесь, если назначение проекта за время его разработки изменится. Такая ситуация возникает довольно часто, и если изменения не окажутся для вас сюрпризом и вы к ним будете готовы, то всегда окажетесь в выигрышной позиции. На всякий случай убедитесь, что в вашем контракте (если таковой имеется) учтена возможность изменения спецификаций проекта.

Проектирование приложения с учетом потребностей пользователя

Определив потребности конечных пользователей, вы можете почувствовать желание погрузиться в работу, т.е. прийти на помощь тому, кто страдает от нерешенной проблемы. И все же наберитесь терпения. Строители не возводят дом без чертежей, и вам не следует разрабатывать приложение электронных таблиц без предварительного плана. Конечно, правильность плана зависит от области действия проекта и привычного вам стиля работы, однако подумайте хотя бы *некоторое* время над тем, что же вы собираетесь делать и чего хотите достичь.

Прежде чем закатать рукава и сесть за клавиатуру, обдумайте разные способы, которые помогут приблизить решение проблемы. Вот здесь как раз и окупится глубокое познание возможностей Excel! Всегда лучше обойти незнакомые закоулки стороной, чем блуждать в них.

Если вы попросите десять "гуру" Excel разработать приложение на основе очень точных спецификаций, то, скорее всего, получите десять разных реализаций проекта, которые все как одна будут соответствовать предложенным спецификациям. Среди этих решений одни наверняка будут лучше, чем другие, поскольку Excel позволяет выполнить одно и то же задание несколькими способами. И если вы досконально изучите эту программу, то будете иметь достаточно хорошее представление о методах, которые имеются в вашем распоряжении, и сможете выбрать самый оптимальный способ реализации текущего проекта. Часто именно благодаря только творческому подходу рождается наиболее эффективный (и значительно превосходящий другие) способ.

Итак, на начальном этапе планирования проекта вам придется обдумать следующие вопросы.

- **Файловая структура.** Подумайте, что вы будете использовать: одну рабочую книгу с множеством листов, несколько однолистных рабочих книг или файл шаблона.
- **Структура данных.** Обязательно учтите структуру данных, которые будут использоваться в приложении. В том числе необходимо решить, использовать ли файлы внешних баз данных или хранить всю информацию в рабочих листах.
- **Формулы или VBA.** Обдумайте, что требуется для вычислений: формулы или процедуры VBA? Каждый из представленных вариантов имеет свои достоинства и недостатки.
- **Надстройка или файл рабочей книги.** В некоторых случаях наилучшим вариантом конечного продукта является надстройка, хотя не исключено применение надстройки вместе со стандартной рабочей книгой.
- **Версия Excel.** Где будет запускаться разработанное Excel-приложение? Только в среде Excel 2010? В среде Excel 2007? Или в Excel 2000/2002? А как насчет Excel 97, Excel 95 либо Excel 5? Будет ли приложение работать на платформе Macintosh? Это очень важные вопросы, поскольку в каждой следующей версии Excel появляются новые свойства, отсутствовавшие в предыдущих версиях. Новый пользовательский интерфейс вызывает больше трудностей, чем создание приложений, работающих с прежними версиями.
- **Обработка ошибок.** Для приложений немаловажным вопросом является обработка ошибок. Определите, как ваше приложение будет "отлавливать" ошибки и что оно будет с ними делать. Например, если ваше приложение применяет форматирование к активному рабочему листу, то необходимо предусмотреть случай, когда активным будет лист диаграммы.
- **Использование специальных возможностей.** Если в вашем приложении будет суммироваться большое количество данных, то подумайте над использованием такого средства Excel, как сводные таблицы. Вам также потребуется такая возможность Excel, как проверка данных, чтобы тестировать вводимые данные.
- **Вопросы производительности.** Решить вопрос увеличения производительности и эффективности своего приложения следует еще на стадии проектирования, а не тогда, когда приложение создано и от пользователей поступают жалобы.
- **Уровень безопасности.** В Excel предусмотрено несколько вариантов защиты, которые призваны предотвратить доступ к определенным элементам рабочей книги. Например, вы можете блокировать ячейки, чтобы нельзя было изменить находящиеся в них формулы, или назначить пароль, чтобы неавторизованные пользователи не смогли просматривать определенные файлы или получать доступ к ним. Вы облегчите свою работу, если заблаговременно и точно определите, какой именно компонент нуждается в защите и каким должен быть уровень этой защиты.

Примечание

Не стоит абсолютно доверять средствам защиты данных в Excel. Если нужно обеспечить действительно эффективную защиту, вряд ли для этого подойдет Excel.

На данном этапе вам, возможно, придется иметь дело со многими другими трудностями разработки приложения. Важно рассмотреть все возможности и не начинать реализовывать первое решение, которое придет вам в голову.

Кроме того, при разработке приложения не забывайте о его возможных изменениях. Вы добьетесь больших успехов, если ваше приложение будет как можно более универсальным. Например, не создавайте процедуру, которая работает только с определенным диапазоном ячеек. Пусть ваша процедура в виде аргумента принимает любой диапазон данных. Когда придет время изменений в проекте, такая возможность существенно облегчит процесс редактирования приложения. Кроме того, вы, возможно, увидите, что текущий проект в определенной мере напоминает другой проект. Поэтому при планировании всегда помните о такой возможности, как повторное использование одних и тех же структур.

Обучение в процессе разработки

Теперь несколько слов о реальном положении дел. Excel — программа изменчивая. Период между ее обновлениями составляет от 18 до 24 месяцев. Это означает, что в вашем распоряжении менее двух лет, чтобы справиться с текущими инновациями, иначе вам придется бороться не только с ними, но и с другими нововведениями.

Программа Excel 5, которая подарила нам VBA, стала для разработчиков Excel "изменением парадигмы" управления данными. Ранее тысячи людей зарабатывали себе на жизнь, создавая приложения Excel, которые в основном писались на макроязыке XLM, представленном в версиях Excel 2, 3 и 4. Начиная с Excel 5 стали доступными десятки новых инструментов, которые активно используются разработчиками.

С появлением программы Excel 97 разработчики столкнулись с еще одной "сменой парадигмы". В этой версии появился новый формат файлов, редактор языка Visual Basic Editor (VBE) и пользовательские формы, которые пришли на смену диалоговым листам. В Excel 2000, 2002 и 2003 появилась новые возможности, которые были уже не столь радикальными, как в предыдущих версиях.

Программа Excel 2007 — это огромный шаг вперед по сравнению с предыдущими версиями, и основная проблема связана с появлением нового "ленточного" интерфейса пользователя. Ранее создание пользовательских меню и панелей инструментов было относительно простым и могло выполняться исключительно средствами VBA. Теперь же изменение ленты требует дополнительной работы, причем не только с помощью VBA. Кроме того, следует учитывать новые форматы файлов. Исходя из этого, я рекомендую создавать две версии приложения: для Excel 2007/2010, а также для Excel 2003 и более ранних версий.

Язык VBA изучить несложно, но потребуется время, чтобы почувствовать себя комфортно: для совершенного овладения этим языком придется приложить немало усилий. Развитие VBA продолжается. Следовательно, не удивительно, что, разрабатывая с помощью VBA приложения, вы одновременно изучаете этот язык. Более того, изучить VBA невозможно, если вплотную не заниматься разработкой приложений. Намного проще изучить VBA, имея проект, для реализации которого требуется исключительно этот язык. Изучение VBA ради "любви к искусству" вряд ли приведет к серьезным результатам.

Опыт показывает, что нельзя при решении проблемы целиком полагаться на конечных пользователей. Предположим, руководителю отдела требуется приложение, генерирующее текстовые файлы, которые будут импортироваться в другое приложение. В данном случае главное — не путать потребности пользователя с искомым решением. Пользователю необходим совместный доступ к данным. А использование промежуточного

текстового файла является всего лишь частным решением этой проблемы. Ведь могут быть и другие частные решения, например прямая передача информации с помощью DDE или OLE. Иначе говоря, нужно запретить пользователю представлять его проблему в виде конечной задачи. Определить наиболее удачное решение *общей* проблемы — это уже *ваша* работа.

Определение удобного пользовательского интерфейса

Разрабатывая электронные таблицы, которые будут применяться другими людьми, следует обратить особое внимание на пользовательский интерфейс. *Пользовательский интерфейс* — это метод взаимодействия пользователя с приложением и вызова макросов.

С появлением Excel 2010 (и Excel 2007) некоторые из применявшихся ранее решений уже не годятся. Так, например, уже не применяются пользовательские меню и панели инструментов. Современным разработчикам приходится учиться работать с лентой.

Вниманию разработчиков Excel предлагается ряд свойств, имеющих отношение к разработке пользовательского интерфейса:

- настройка ленты;
- настройка контекстного меню;
- "быстрые" клавиши;
- создание пользовательских диалоговых окон (пользовательских форм);
- размещение непосредственно на рабочем листе элементов управления (например, `ListBox` либо `CommandButton`).

Эти возможности кратко рассмотрены в следующих разделах, а более подробно — в следующих главах.

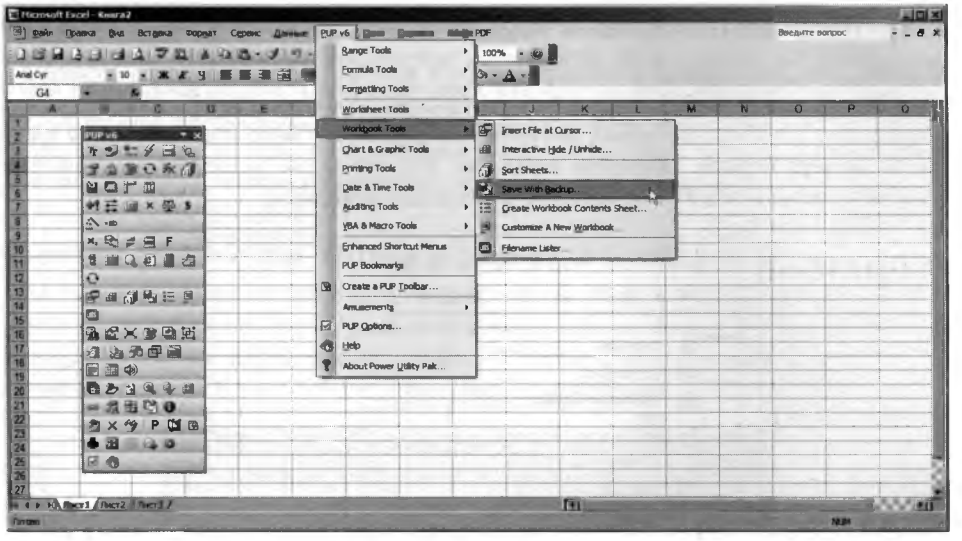
Меню и панели инструментов

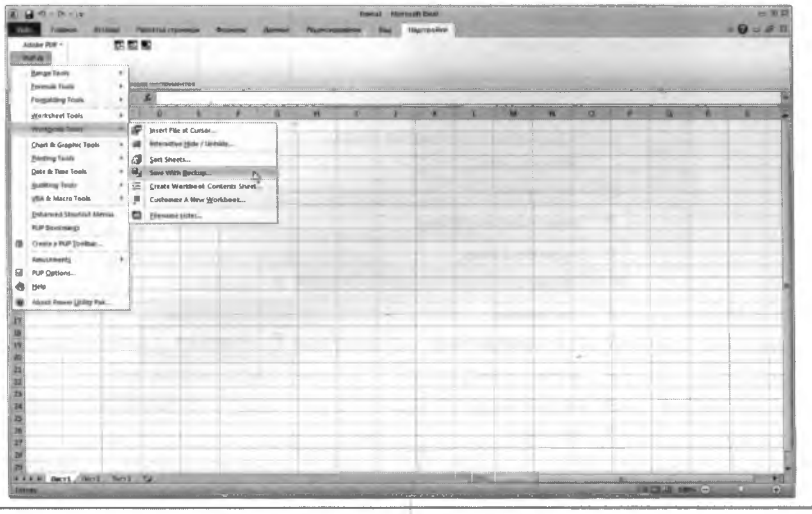
В Excel 2010 поддерживаются пользовательские меню и панели инструментов, хотя и при разработке этих элементов пользовательского интерфейса у пользователя могут возникать определенные проблемы.

На следующем рисунке показаны пользовательское меню и панель инструментов, которые отображены в Excel 2003. Эти элементы были созданы с помощью надстройки Power Utility Pak. Каждый элемент меню и панели инструментов вызывает макрос.

Когда надстройка Power Utility Pak установлена в Excel 2010 (см. следующую иллюстрацию), пользовательское меню появляется в группе Надстройки⇨Команды меню (Add-Ins⇨Menu Commands), а пользовательская панель инструментов попадает в группу Надстройки⇨Пользовательские панели инструментов (Add-Ins⇨Custom Toolbars). Обратите внимание, что панели инструментов нельзя перемещать, а также невозможно изменять их размеры. Эти группы ленты отображают дополнения к меню и панели инструментов для всех загруженных приложений либо надстроек. Кнопки меню и панелей инструментов вполне функциональны, хотя их использование приводит к нарушению исходной парадигмы интерфейса пользователя Excel 2010.

Для решения этой проблемы я разработал новую версию утилиты PUP, предназначенную для Excel 2007 и более поздних версий.





Настройка ленты

Ленточный интерфейс, появившийся в Excel 2007, представлял собой революционный прорыв в области разработки пользовательских интерфейсов. Причем разработчики получили возможность контролировать внешний вид и функции ленты. С появлением версии Excel 2010 пользователи также получили возможность изменять ленту, хотя модификация интерфейса пользователя с помощью кода является непростой задачей.

Перекрестная ссылка

Дополнительные сведения о работе с лентой можно найти в главе 22.

Настройка контекстных меню

В Excel 2010, как и раньше, поддерживаются контекстные меню, разрабатываемые на языке VBA, которые вызываются щелчком правой кнопки мыши. На рис. 6.1 показано настраиваемое контекстное меню, которое отображается после щелчка правой кнопкой мыши на номере строки. Обратите внимание, что это меню включает дополнительные пункты (помечены буквой "P"), которые изначально невидимы.

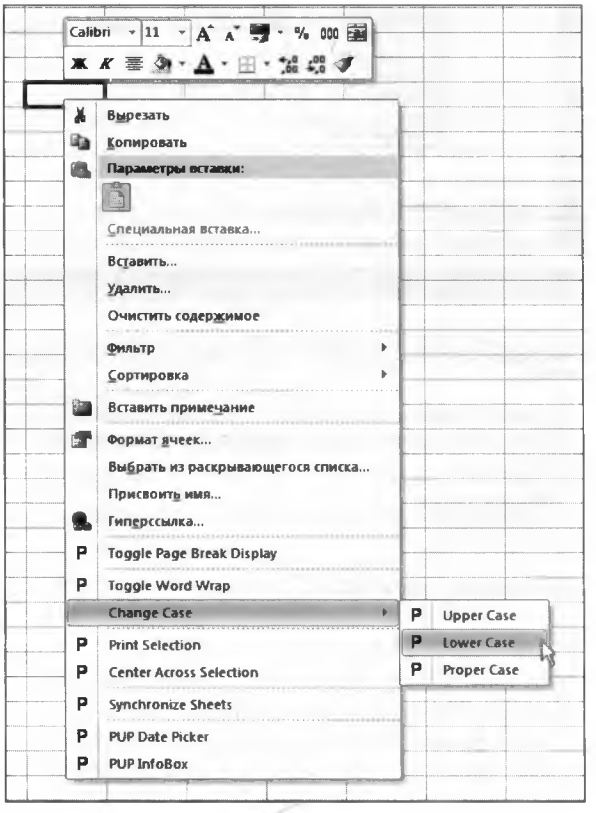


Рис. 6.1. Пример настраиваемого контекстного меню

Перекрестная ссылка

В главе 23 описывается работа с контекстным меню с помощью VBA.

Комбинации клавиш

В распоряжении разработчика Excel имеется ряд комбинаций клавиш. В частности, макросу можно назначить клавишу <Ctrl> (либо комбинацию клавиш <Shift+Ctrl>). После нажатия этих клавиш макрос вызывается на выполнение.

Имейте в виду, что в этом случае могут быть свои "подводные камни". Для того чтобы их избежать, следует объяснить пользователю назначение этих клавиш. Также при выборе комбинации клавиш проверяйте, не используются ли они еще где-нибудь. Ком-

бинация клавиш, назначаемая макросу, имеет больший приоритет, чем встроенные комбинации клавиш. Например, рассмотрим встроенную в Excel комбинацию клавиш <Ctrl+S>, используемую для сохранения текущего файла. Если эту комбинацию использовать для вызова макроса, вы не сможете сохранить файл нажатием клавиш <Ctrl+S>. Обратите внимание, что комбинации клавиш чувствительны к регистру символов, поэтому в данном случае можно использовать клавиши <Ctrl+Shift+S>.

Создание пользовательских диалоговых окон

Если вы какое-то время работали в Excel, то, несомненно, знакомы с диалоговыми окнами. Пользовательские диалоговые окна играют важную роль в тех интерфейсах, которые вы разрабатываете для своих приложений. Пример пользовательского диалогового окна показан на рис. 6.2.

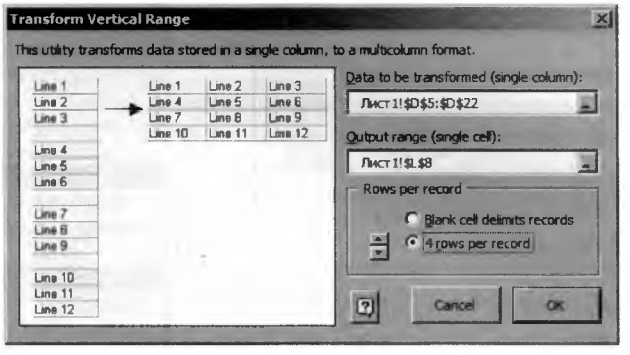


Рис. 6.2. Диалоговое окно, созданное с помощью свойства пользовательских форм Excel

С помощью пользовательского диалогового окна (UserForm) пользователь может ввести те или иные данные, получить выбранные им варианты или предпочтения, а также задать ход выполнения всего приложения. Создавать и редактировать пользовательские диалоговые окна вы можете в редакторе Visual Basic (Visual Basic Editor — VBE). Элементы, из которых состоит диалоговое окно, называются *элементами управления* (к ним относятся кнопки, раскрывающиеся списки, флажки и т.д.). Они также называются *элементами управления ActiveX*. Можно пользоваться не только стандартным набором таких элементов, который поддерживается в Excel. Вы вправе применять элементы управления, которые созданы сторонними производителями.

Добавив элемент управления в диалоговое окно, можно связать его с ячейкой рабочей таблицы, поэтому ему не потребуется макрос для управления (если не считать простого макроса, с помощью которого отображается само диалоговое окно). Связать элемент управления с ячейкой несложно, однако такой способ приема данных, вводимых пользователем в диалоговое окно, не всегда является лучшим. Гораздо чаще для работы с разработанными вами диалоговыми окнами придется создавать VBA-макросы.

Перекрестная ссылка

Подробнее пользовательские формы будут рассмотрены в части IV.

Использование элементов управления ActiveX на рабочем листе

В Excel элементы управления ActiveX, предназначенные для пользовательских форм, можно вставлять и на графический слой (невидимый слой в верхней части листа, на котором находятся рисунки, диаграммы и другие объекты). На рис. 6.3 представлена простая модель рабочего листа с несколькими элементами управления пользовательских форм. На рабочем листе находятся следующие элементы управления: флажок (`Checkbox`), полоса прокрутки (`ScrollBar`), а также два набора переключателей (`OptionButtons`). Эта рабочая книга не использует макросы, а элементы управления связаны с ячейками рабочего листа.

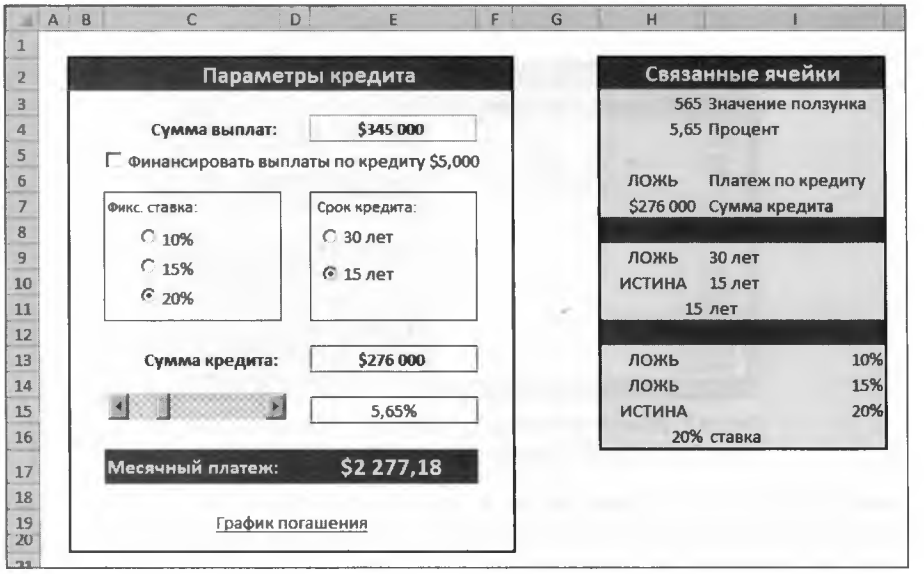


Рис. 6.3. Можно добавить элементы управления диалоговым окном в рабочие листы, а затем связать их с ячейками

Компакт-диск

Эту рабочую книгу можно найти на прилагаемом компакт-диске (файл `worksheet controls.xlsx`).

Возможно, самым распространенным элементом управления является `CommandButton`. Сами по себе кнопки не выполняют никаких функций, но каждой из них вы можете назначить макрос.

Используя элементы управления непосредственно на рабочем листе, можно не создавать пользовательские диалоговые окна. Часто вставка в рабочий лист нескольких элементов управления ActiveX значительно упрощает работу с электронной таблицей. В результате пользователь сможет выбирать то, что ему нужно, работая со знакомыми элементами управления, а не вводя значения в ячейки.

Для получения доступа к элементам управления используется команда **Разработчик⇨Элементы управления⇨Вставить** (Developer⇨Controls⇨Insert) (рис. 6.4). Если вклад-

ка Разработчик не отображается на ленте, устраните эту проблему с помощью раздела Настройка ленты, находящегося в диалоговом окне Параметры Excel.

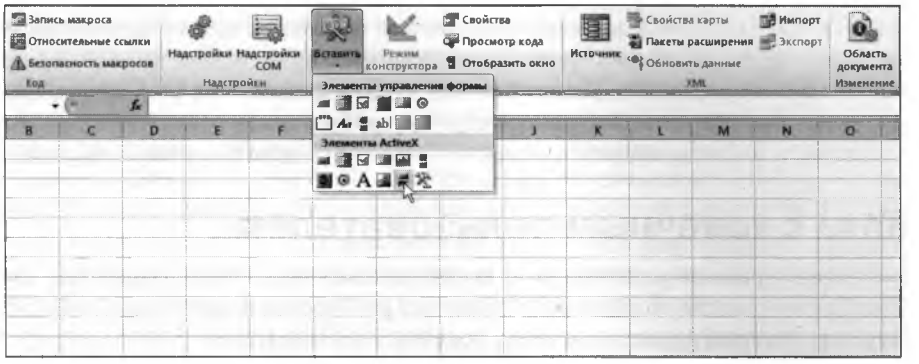


Рис. 6.4. Элементы управления на рабочем листе

Все элементы управления делятся на две группы: элементы управления формами и элементы управления ActiveX. Оба набора элементов управления имеют свои преимущества и недостатки. В общем случае элементы управления формами проще в применении, но зато элементы управления ActiveX более гибкие. Информация об этих классах элементов управления представлена в табл. 6.1.

Таблица 6.1. Элементы управления формами и элементы управления ActiveX

	Элементы управления ActiveX	Элементы управления формами
Версии Excel	97, 2000, 2002, 2003, 2007, 2010	5, 95, 97, 2000, 2002, 2003, 2007, 2010
Элементы управления	`CheckBox`, `TextBox`, `CommandButton`, `OptionButton`, `ListBox`, `ComboBox`, `ToggleButton`, `SpinButton`, `ScrollBar`, `Label`, `Image` (могут быть и другие)	`GroupBox`, `Button`, `CheckBox`, `OptionButton`, `ListBox`, `DropDown` (`ComboBox`), `ScrollBar`, `Spinner`
Хранение кода макросов	В любом стандартном VBA-модуле листа	В модуле кода
Имя макроса	Соответствует имени элемента управления (например, `Command_Button1_Click`)	Любое указанное вами имя
Чему соответствует	Элементам управления пользовательских форм	Элементам управления диалоговых листов (в версиях, предшествующих Excel 97)
Настройка	В широком объеме с помощью панели Properties	Минимальная
Есть ли реакция на события	Только на события `Click` (Щелчок) и `Change` (Изменение)	Да

Разработка собственно приложения

Когда вы определите потребности пользователя, подберете тип проекта, который должен удовлетворить их запросы, и решите, какие компоненты необходимо применить

в пользовательском интерфейсе, настанет время завершить подготовительные операции и приступить к созданию самого приложения. Этот этап, конечно же, занимает значительную часть времени, которое тратится на реализацию проекта.

Каким образом вы будете создавать приложение, зависит от вашего персонального стиля работы и от природы самого приложения. Если ваше приложение не является простой рабочей книгой, содержащей шаблоны бланков для заполнения, то в нем, скорее всего, будут использоваться макросы. Написание макросов требует много времени и сил. Не думайте, что создавать макросы невероятно сложно — трудно создавать *хорошие* макросы.

Работа с конечным пользователем

В настоящем разделе речь пойдет о весьма важных проблемах разработки, которые дают о себе знать, когда приложение становится работоспособным и приближается время размещать готовый проект в системах конечных пользователей.

Тестирование приложения

Сколько раз коммерческое приложение "отказывает" в самый неподходящий момент? Скорее всего, проблема вызвана недостаточно качественным тестированием, в результате которого были обнаружены не все ошибки. Конечно, ошибки есть во всех коммерческих программах, просто в самых лучших программах их заметить труднее. Вы часто будете сталкиваться с ситуацией, когда необходимо обойти ошибки, не отслеженные в самой программе Excel, иначе ваше приложение так и не заработает.

Создав приложение, не забудьте его протестировать. Это один из самых важных этапов. Нередко на тестирование и отладку приложения уходит столько же времени, сколько на создание его исходного варианта. В конце концов, пишете вы процедуру VBA или создаете формулы рабочего листа, вам все равно захочется убедиться, что приложение работает именно так, как предполагалось.

Склонность к ошибкам имеют не только стандартные компилированные приложения. То же самое можно сказать и о приложениях электронных таблиц. *Ошибка* обычно определяется следующим образом: (1) это нечто такое, что при выполнении программы (или приложения) происходит, но происходить не должно; (2) это нечто такое, что не происходит, но происходить как раз обязано. Оба вида ошибок одинаково опасны. Поэтому значительную часть времени, которое уйдет на разработку, вы должны выделить на тестирование приложения во всех возможных условиях и на исправление любых возникших проблем. К сожалению, иногда не только вы виноваты в проблемах. Свою лепту вносит и сама программа Excel (см. врезку "Ошибки? В Excel?").

Ошибки? В Excel?

Вы, возможно, думаете, что такой продукт, как Excel, используемый миллионами людей по всему миру, не должен содержать ошибок (по крайней мере грубых). Увы, это не так. Программа Excel настолько сложна, что вполне естественно ожидать от нее неадекватных действий. И конечно же, подобные действия вызываются разного рода ошибками.

Создать такой программный продукт, как Excel, — задача не из легких даже для компании Microsoft с ее, на первый взгляд, неограниченными ресурсами. При выпуске программы нельзя избежать компромиссов. Общеизвестно, что крупные производители выпускают свои приложения, прекрасно осознавая, что в тех содержатся ошибки. Большинство этих ошибок настолько незначительны, что на них вообще можно не обращать

внимания. Конечно, компании — производители программ могли бы повременить с выпуском своих продуктов на пару месяцев и исправить большинство ошибок. Но программы, как и многие другие товары, являются "скоропортящимися" продуктами. Прибыли от задержанного по выпуску продукта часто меньше израсходованных на него средств. И хотя в программе Excel имеются ошибки, большинство ее пользователей никогда с ними не столкнется.

В данной книге отмечены важные проблемы, связанные с использованием Excel. Кроме них, вы, конечно же, найдете и другие ошибки. Некоторые проблемы характерны только для конкретной версии Excel и в определенной аппаратно программной конфигурации. Это наихудшие из ошибок, поскольку их распознать весьма непросто.

Так как же быть разработчику приложений? Именно так, как призывает доктрина *обхода ошибок*. Если то, что вы пытаетесь делать, не работает, хотя все говорит о том, что *должно* работать, самое время перейти к плану Б. Обидно? Конечно! Потеря времени? Несомненно! Такова судьба разработчика...

Вероятно, не стоит напоминать о необходимости тщательного тестирования любых электронных таблиц, которые вы разрабатываете для других. Учитывая особенности пользовательской аудитории своего приложения, вы можете сделать его отказоустойчивым. Другими словами, попробуйте прогнозировать все возможные ошибки и недочеты, которые потенциально могут допускаться пользователями, и приложите усилия, чтобы их избежать (или хотя бы красиво скрыть). Это не только поможет конечному пользователю, но и позволит не запятнать вашей репутации.

Конечно, не в ваших силах прогнозировать все возможности, однако создаваемые макросы должны обрабатывать основные виды ошибок. Например, пользователь вместо числового значения ввел текстовое или пытается запустить ваш макрос, когда рабочая книга еще не открыта. Он может скрыть диалоговое окно, так ничего и не настроив, или нажать комбинацию клавиш <Ctrl+F6> и перейти к следующему окну. Что делать в этих ситуациях? Когда вы приобретете достаточный опыт, то такого рода вопросы станут для вас обязательными при тестировании приложений, и вы будете отвечать на них без лишних раздумий.

А как же бета-тестирование?

Производители программ обычно предусматривают для своих новых продуктов процедуры тщательного тестирования. После интенсивного внутреннего тестирования выпуску продукта на рынок обычно предшествует его тестирование группой заинтересованных пользователей. Подобная операция приобрела название *бета-тестирования*. На этом этапе часто обнаруживаются новые проблемы, обычно устраняемые ко времени выпуска конечного продукта.

Если вы разрабатываете в Excel приложение, которым будет пользоваться большое количество человек, то, возможно, будет нелишним провести его бета-тестирование. Такое тестирование позволит запустить созданное приложение в предназначенном для него программном окружении на, как правило, разном оборудовании и именно теми пользователями, для которых оно предназначено.

Период бета-тестирования начинается после завершения самостоятельного тестирования приложения и перед распространением программного продукта среди пользователей. Определите группу пользователей, которые будут помогать вам тестировать продукт. Наилучшие результаты бета-тестирования достигаются тогда, когда бета-тестеры получают в свое распоряжение абсолютно все, что входит в комплект поставки приложения: программную документацию, программу установки, компьютерную спра-

вочную систему и т.д. Узнать результаты бета-тестирования можно путем личного общения, анкетирования или по телефону.

Прежде чем отважиться на распространение своего продукта среди пользователей, вы наверняка столкнетесь с проблемами, для устранения которых потребуется внести дополнительные исправления или выполнить корректировку. Конечно, на бета-тестирование требуется дополнительное время, и не все проекты (и разработчики) могут позволить себе подобную роскошь.

Как сделать приложение отказоустойчивым

Нарушить структуру электронной таблицы очень легко. Часто удаление одной важной формулы или ключевого значения вызывает ошибки во всей таблице и даже в других зависимых таблицах. Более того, если поврежденную таблицу сохранить, то на диске она заменит правильную копию. И если резервное копирование приложения вами не выполнялось, то его пользователь будет *очень* расстроен, в чем, скорее всего, обвинит именно *вас*.

Теперь понятно, зачем устанавливается дополнительная защита приложения, прежде чем пользователи (особенно начинающие) приступят к его применению для решения конкретных задач. В Excel для защиты рабочих листов и их частей можно использовать несколько способов.

- **Блокирование определенных ячеек.** Можно заблокировать определенные ячейки, чтобы предотвратить их изменение. (Задается с помощью вкладки Защита (Protection) диалогового окна Формат ячеек (Format Cells).) Блокировка вступает в силу только после выбора команды Рецензирование⇨Изменения⇨Защитить лист (Review⇨Changes⇨Protect Sheet). В диалоговом окне Защита листа (Protect Sheet) находится ряд параметров, определяющих действия, которые выполняются по отношению к защищенному листу (рис. 6.5).
- **Сокрытие формул для некоторых ячеек.** Если нужно скрыть формулы для определенных ячеек, воспользуйтесь вкладкой Защита (Protection) в диалоговом окне Формат ячеек (Format Cells), после чего эти формулы не смогут видеть другие пользователи. И опять же, эта команда действует только в том случае, когда документ защищен путем применения к нему команды Рецензирование⇨Изменения⇨Защитить лист (Review⇨Changes⇨Protect Sheet).
- **Защита всей рабочей книги.** Можно защитить всю рабочую книгу — ее структуру, расположение окон и их размеры либо то или другое. Для этого используйте команду Рецензирование⇨Изменения⇨Защитить книгу (Review⇨Changes⇨Protect Workbook).

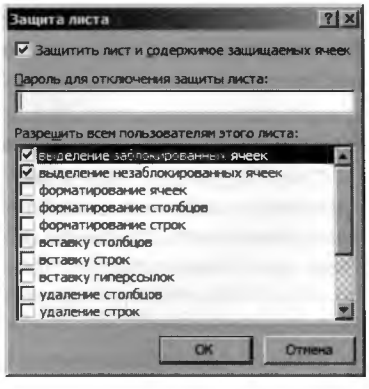


Рис. 6.5. В диалоговом окне Защита листа (Protect Sheet) указывается, что можно и чего нельзя делать пользователям

- **Блокирование объектов на рабочем листе.** Перейдите на вкладку Свойства (Properties) в диалоговом окне Размер и свойства (Size and Properties) для блокирования объектов (например, фигур), чтобы предотвратить их изменение либо перемещение. Для получения доступа к этому диалоговому окну выделите объект

и щелкните на значке открытия диалогового окна в группе Средства рисования⇨Формат⇨Размер (Drawing Tools⇨Format⇨Size). Блокирование объектов дает эффект в том случае, когда документ защищен путем применения команды Рецензирование⇨Изменения⇨Защитить лист (Review⇨Changes⇨Protect Sheet). По умолчанию все объекты заблокированы.

- **Сокрытие строк, столбцов, рабочих листов и документов.** Можно скрывать строки, столбцы, рабочие листы и целые рабочие книги. Благодаря этому рабочий лист приобретает аккуратный вид, а защищенные области скрываются от любопытных глаз.
- **Выбор для рабочей книги Excel режима "только чтение".** Можно перевести рабочую книгу Excel в режим "только чтение" (и воспользоваться паролем), что позволит гарантировать вероятность перезаписи файла. Для этого используется диалоговое окно Общие параметры (General Options). Для его открытия выберите команду Файл⇨Сохранить как (File⇨Save As). В диалоговом окне Сохранение документа (Save As) щелкните на кнопке Сервис (Tools) и выберите пункт Общие параметры (General Options).
- **Назначение пароля.** Можно воспользоваться паролем для предотвращения несанкционированного доступа к своим файлам. Назначить пароль можно в диалоговом окне Общие параметры (General Options). Для открытия этого окна выберите команду Файл⇨Сохранить как (File⇨Save As). В диалоговом окне Сохранение документа (Save As) щелкните на кнопке Сервис (Tools) и выберите пункт Общие параметры (General Options).
- **Используйте защищенную паролем надстройку.** Можно обратиться к защищенной паролем надстройке, которая предотвращает изменение всего, что есть на рабочих листах.

Защита данных паролем в Excel

Компания Microsoft никогда не рассматривала Excel в качестве полностью защищенной программы. И тому есть определенная причина — взломать систему паролей Excel на самом деле совершенно несложно. Существует несколько коммерческих программ, с помощью которых взлом паролей не представляет особого труда. Программа Excel 2002 и более поздние версии снабжены более мощной защитой (по сравнению с предыдущими версиями), но все же этого недостаточно. Не стоит считать защиту в Excel абсолютно надежной. Конечно, она достаточна, чтобы противодействовать случайному пользователю, но если кому-то действительно захочется взломать пароль, то он, скорее всего, добьется своего.

Создание привлекательных и интуитивно понятных приложений

Если вам приходилось использовать несколько программных пакетов, то вы, несомненно, видели плохо разработанные пользовательские интерфейсы, сложные в применении программы и просто отвратительные диалоговые окна. Разрабатывая электронные таблицы для других пользователей, не забывайте уделить особое внимание внешнему виду своего приложения.

У конечных пользователей первая реакция на приложение связана исключительно с внешним видом программы. Это в полной мере относится и к приложениям, которые вы разрабатываете с помощью Excel. У каждого, впрочем, свое представление о красоте. И если вы вышли на серьезный уровень профессиональной деятельности, то подумайте над тем, чтобы к разработке приложения привлечь человека с хорошим вкусом.

Хорошие новости заключаются в том, что начиная с Excel 2007 создание профессионально выглядящих рабочих листов не представляет особого труда. Просто воспользуйтесь предварительно определенными стилями для ячеек, и вы получите достойный результат. А для применения новой темы, которая полностью изменяет внешний вид рабочей книги, достаточно один раз щелкнуть мышью. К сожалению, версия Excel 2010 не добавила ничего нового в область разработки пользовательских форм, поэтому придется пользоваться старыми возможностями.

Следует дополнительно уделить внимание вопросам дизайна и эстетики — и пользователи обязательно оценят ваше умение разрабатывать не только удобный, но и красивый интерфейс. Само же приложение будет выглядеть утонченно и профессионально. Хороший внешний вид приложения говорит о том, что разработчик настолько заботился о своем продукте, что не пожалел времени и усилий на создание интерфейса. Итак, в процессе разработки пользовательского интерфейса учитывайте следующие моменты.

- **Добивайтесь единообразия.** Разрабатывая, например, диалоговые окна, старайтесь по возможности следовать внешнему виду диалоговых окон Excel. Соблюдайте единообразие в формате, шрифтах, размере текста и цветах.
- **Выбирайте простые решения.** Распространенной ошибкой разработчиков является то, что они пытаются вывести на экран или в диалоговом окне как можно больше информации. Добиваясь этого, следуйте эмпирическому правилу, согласно которому информацию необходимо выдавать порциями — не более одного либо двух блоков за раз.
- **Разбивайте окна ввода.** Если для получения информации от пользователя вы используете диалоговое окно, то подумайте над тем, чтобы разбить его на несколько окон, каждое из которых будет менее загроможденным. Если вы все же решили использовать сложное диалоговое окно, то, чтобы его разбить, можете воспользоваться элементом управления `MultiPage` (Множество вкладок), который помогает создать знакомое вам диалоговое окно с несколькими вкладками.
- **Не переусердствуйте с цветом.** Цвета используйте осторожно, потому что с ними легко переусердствовать.
- **Отслеживайте шрифты и графику.** Уделите особое внимание числовым форматам, начертанию, размерам шрифтов, а также границам.

Эстетика — понятие довольно субъективное, но если вы сомневаетесь в своих способностях, то постарайтесь добиться как минимум простоты и ясности интерфейса приложения.

Примечание

В версиях, предшествующих Excel 2007, использовалась палитра из 56 цветов. Теперь это ограничение в прошлом, и в Excel 2010 поддерживается свыше 16 миллионов цветов.

Создание пользовательской справочной системы

Что же касается пользовательской документации, то используется преимущественно два варианта: документация в бумажном и электронном виде. Электронная справочная система — это стандартный компонент Windows-приложений. Ваши Excel-приложения не исключения — они могут иметь электронную справочную систему и даже ее контекстную разновидность. На разработку электронной справочной системы требуется много времени и усилий, но в большом проекте они, конечно же, с лихвой окупятся. На рис. 6.6 показан пример пользовательской справочной системы в скомпилированном HTML-формате.

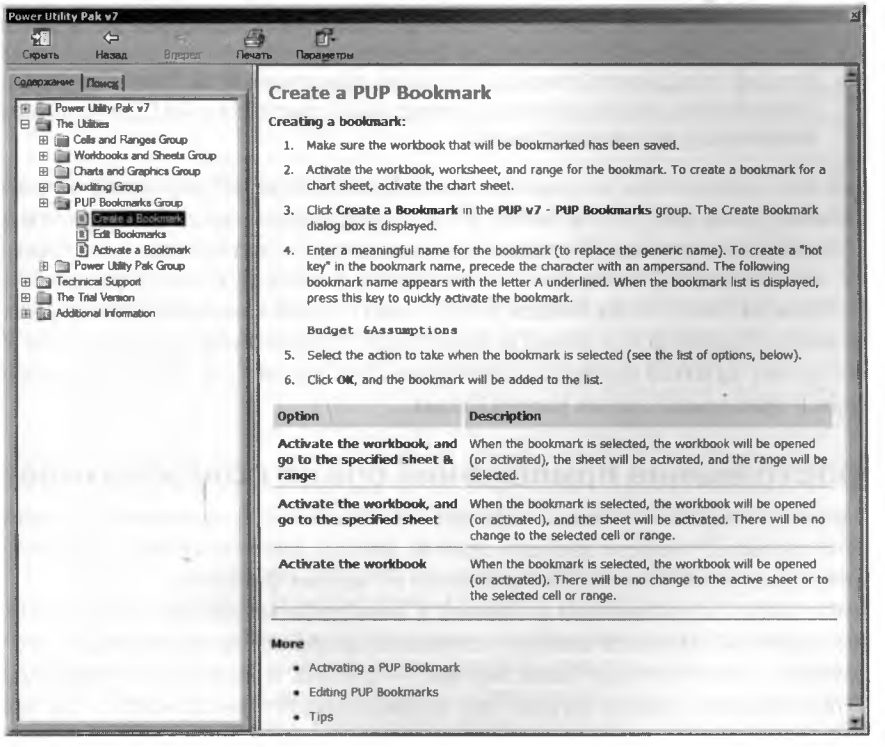


Рис. 6.6. Пример файла пользовательской справки для надстройки Excel

Кроме указанных выше рекомендаций, следует уделить особое внимание технической поддержке приложения в будущем. Иными словами, кому будут звонить пользователи при возникновении проблем с его использованием? Если вы не готовы отвечать на такие вопросы, то найдите того, кто будет этим заниматься. В большинстве случаев вам захочется отвечать только на сложные "технологические" вопросы, касающиеся исправления ошибок программного кода.

Перекрестная ссылка

В главе 24 рассматриваются альтернативные варианты справочных систем для ваших приложений.

Документирование усилий, затраченных на разработку

Собрать приложение электронных таблиц в единое целое — это одна задача, а сделать приложение понятным для других людей — совсем другая. Как и в традиционном программировании, в данном случае важно тщательно документировать свою работу. Созданная документация пригодится тогда, когда вам потребуется доработать приложение. Она также понадобится тому, кому вы в дальнейшем передадите свое приложение.

Совет

При документировании проекта придется учесть следующий момент. Например, если вас пригласили для разработки приложения Excel, то вам, видимо, не захочется, тщательно документируя все и вся, делиться всеми секретами созданного тяжелым трудом приложения. В таком случае следует подготовить два варианта документации: сокращенный (для пользователей) и полный (для себя любимого).

Как же документировать приложение электронных таблиц? Информацию можно хранить в рабочей книге или другом файле. Если хотите, можете представить документацию в виде бумажного документа. Возможно, проще всего — использовать отдельный рабочий лист, чтобы хранить в нем свои комментарии и основную информацию о проекте. Что касается кода VBA, то вы можете использовать в нем комментарии (ведь код, перед которым введен апостроф, все равно игнорируется). Элегантный фрагмент кода VBA сегодня вам может казаться предельно очевидным, но, изучив его через пару месяцев, вы запутаетесь в приводимых ранее рассуждениях.

Распространение приложения среди пользователей

Вы завершили проект и готовы передать его конечному пользователю. Каким образом вы это сделаете? Можете выбрать один из многих доступных способов распространения программных продуктов. Выбор зависит от многих факторов.

Создайте диск с приложением и внесите в него простые инструкции по использованию. Вам, конечно, захочется самому установить созданное приложение, но это не всегда возможно. Существует еще один вариант — создать официальную программу установки. Эта программа предназначена для автоматического выполнения всех необходимых операций по внедрению приложения в компьютер пользователя. Вы можете написать такую программу на традиционном языке программирования, купить общий ее код или написать свою собственную на VBA.

В Excel 2000 и более поздних версиях используется технология Microsoft Authenticode, благодаря которой разработчики могут скреплять свои приложения "цифровой подписью". Эта технология должна помочь конечным пользователям распознавать автора приложения, гарантировать, что проект не изменен сторонними пользователями, и препятствовать распространению макровирусов и другого потенциально разрушительного кода. Чтобы защитить проект цифровой подписью, вначале следует обратиться в специальную организацию по сертификации и получить соответствующий сертификат. (Существует и другая возможность — самостоятельно подписать свой проект, создав уникальный цифровой сертификат). Дополнительная информация на эту тему приведена в электронной справочной системе и на веб-сайте компании Microsoft.

Обновление приложения

Распространив приложение среди пользователей, вы завершили его разработку, не правда ли? И теперь можно отдохнуть, наслаждаясь жизнью и пытаясь забыть о возникших (и решенных вами же) проблемах при разработке приложения. Конечно, в редких случаях приложение действительно можно считать завершенным. Однако чаще случается так, что пользователи приложения не будут полностью им довольны. Всем первоначальным спецификациям оно будет соответствовать, но жизнь ведь не стоит на месте. Когда пользователь встречает правильно работающее приложение, то начинает думать о функциях, которыми можно его пополнить. Поэтому он попросит включить их в приложение. Да, речь идет об *обновлениях*.

Рано или поздно потребуется обновить или переделать приложение. Вот тут вы и оцените, насколько хорошо спроектировали его вначале и полностью ли задокументировали все поддерживаемые операции. Если нет, то... Мы все учимся на собственном опыте.

Почему нет выполняемой версии Excel

Размещая свое приложение в компьютере конечного пользователя, вы хотите быть уверенным в том, что каждый из них имеет лицензионную копию необходимой версии Excel. Передача копии Excel вместе с приложением является незаконной. У вас, возможно, возникнет вопрос "Почему же компания Microsoft не создала выполняемую версию Excel?" Выполняемая версия — это программа, которая может загружать файлы, но не создавать их. Имея такую версию, пользователи не нуждались бы в копии Excel для запуска вашего приложения. (Так часто бывает с приложениями баз данных.)

Трудно найти ясное и убедительное объяснение, почему Microsoft не распространяет выполняемой версии Excel и почему подобной маркетинговой концепции придерживаются другие производители процессоров электронных таблиц. Наиболее вероятная причина заключается в следующем: производители боятся, что из-за выпуска выполняемой версии уменьшатся объемы продаж полных версий программ. Или, возможно, разработка выполняемой версии потребует дополнительного перепрограммирования имеющегося кода, что никогда не окупится на современном рынке программных продуктов.

Рассуждение на тему... Компания Microsoft настойчиво предлагает программу *просмотра* файлов Excel. Эта утилита обеспечивает возможность просмотра файлов Excel без установки копии Excel. Правда, макросы в программе просмотра не выполняются. Копию бесплатной программы просмотра можно получить на веб-сайте компании Microsoft (`http://office.microsoft.com/downloads`).

Другие вопросы разработки приложений

При разработке приложения следует помнить и о некоторых других вопросах, особенно тогда, когда вы не знаете точно, кто же будет использовать ваше приложение. Если вы разрабатываете приложение, которое планируется применять в широком кругу пользователей (например, условно-бесплатное приложение), то сложно предугадать, каким образом его будут использовать, в какой системе и вместе с какими программами оно будет запускаться.

Версия Excel, установленная у пользователя

С каждой новой версией Excel вопрос совместимости становится все более актуальным. На время написания этой книги программа Excel 2010 уже продавалась, но во мно-

гих больших корпорациях по-прежнему продолжают использовать Excel 2003 и даже более ранние версии.

К сожалению, нет никакой гарантии, что приложение, разработанное, скажем, для Excel 2000, будет хорошо выполняться в последующих версиях Excel. Если необходимо, чтобы приложение работало с различными версиями Excel, то вам потребуется работать с самой старой версией программы, а затем протестировать свой продукт во всех остальных версиях.

Дело еще более усложнится, если учитывать "подверсии" программы Excel. Для исправления тех или иных проблем Microsoft выпускает обновленные версии (SR), пакеты обновлений (SP) и обновления безопасности, предназначенные для исправления обнаруженных проблем. Иногда приложение Excel будет работать некорректно до тех пор, пока вы не установите то или иное обновление.

Перекрестная ссылка

Вопросы совместимости рассматриваются в главе 26.

Трудности, касающиеся поддержки языка

Считайте, что вам повезло, если все конечные пользователи вашего приложения имеют англоязычную версию Excel. Дело в том, что версии этой программы на других языках не всегда полностью совместимы. Поэтому вам придется проводить дополнительное тестирование поддержки приложения в многоязычной среде.

Перекрестная ссылка

Вопросы, связанные с поддержкой языка, рассматриваются в главе 26.

Быстродействие системы

Вероятно, вы — достаточно опытный пользователь компьютера — стараетесь постоянно обновлять его оборудование. Другими словами, у вас довольно производительная система, которая лучше, чем система среднестатистического пользователя. В отдельных случаях вам точно известно, какое аппаратное обеспечение используется конечными пользователями ваших приложений. Если это так, то крайне важно, чтобы вы протестировали приложение в подобной аппаратной среде. Может получиться так, что процедура, которая выполняется в вашей системе почти мгновенно, в другой системе потребует нескольких секунд. А ведь несколько секунд "простоя", потраченных на выполнение простой операции, могут не удовлетворить искушенного современными технологиями пользователя.

Совет

Приобретая все больший опыт работы с VBA, вы обнаружите, что способы выполнения работы и быстрого выполнения работы существенно отличаются. Выработайте привычку скоростного ввода программного кода. (В этом вам помогут остальные главы книги.)

Видеорежимы

Не секрет, что пользователи применяют самые различные видеорежимы. Чаще всего используется разрешение экрана 1024×768, но многие системы настроены на разрешение

800×600. Все более распространенными становятся мониторы с высоким разрешением и даже двухмониторные системы. Но даже если в вашем распоряжении имеется монитор с высоким разрешением, это вовсе не означает, что подобный монитор есть у каждого пользователя ваших приложений.

Разрешение экрана, применяемое пользователями, может быть источником проблем, если ваше приложение основано на информации, отображаемой на единственном экране. Например, если был разработан экран ввода с разрешением 1280×1024, а пользователи применяют разрешение 1024×768, они не смогут просмотреть весь экран без прокрутки либо изменения масштаба.

Кроме того, важно понимать, что восстановленная (т.е. неразвернутая и несвернутая) рабочая книга отображается с предыдущими размерами окна и в предыдущей позиции. Случается и так, что окно, сохраненное при отображении с высоким разрешением, полностью смещается за пределы экрана, когда открывается в VGA-системе.

К сожалению, не существует способа автоматического изменения размера объектов с тем, чтобы они отображались одинаковым образом независимо от разрешения экрана. В некоторых случаях можно изменять масштаб просмотра рабочего листа (с помощью ползунка Масштаб (Zoom), который находится в строке состояния), но этот не всегда удобно. И если вы не знаете, какие разрешения экрана применяются пользователями приложения, то важно, чтобы вы его спроектировали, основываясь на "наименьшем из возможного" — 800×600.

Из главы 10 вы узнаете, что определить применяемое пользователем разрешение можно путем вызова функций Windows API из кода VBA. В некоторых случаях можно настраивать видеоизображение программным путем (в зависимости от используемого разрешения экрана).

Часть III

Visual Basic for Applications

В этой части...

Глава 7

Введение в VBA

В этой главе...

- Основы языка BASIC
- Обзор VBA
- Основы VBA
- Знакомство с редактором Visual Basic
- Работа с Project Explorer
- Работа с окнами кода
- Настройка среды VBE
- Средство записи макросов
- Об объектах и коллекциях
- Свойства и методы
- Объект `Comment`: пример использования
- Некоторые полезные свойства объекта `Application`
- Работа с объектами `Range`
- Что следует знать об объектах

В настоящей главе вы ознакомитесь с языком Visual Basic for Applications, а также с объектами, включенными в Excel.

Основы языка BASIC

Многие опытные программисты не воспринимают идею программирования на BASIC всерьез. Само название (Beginner's All-purpose Symbolic Instruction Code — универсальный символический язык инструкций для начинающих) предполагает, что это непрофессиональный язык. Действительно, BASIC был разработан в начале 1960-х годов и задумывался как наглядное средство преподавания методов программирования студентам колледжей. Довольно быстро он приобрел большую популярность и сейчас поддерживается многими типами компьютеров.

С годами BASIC развивался и улучшался. Например, во многих ранних версиях он был *интерпретируемым* языком. Каждая строка перед выполнением интерпретировалась, чем и была обусловлена низкая скорость обработки кода. В большинстве современных вариантов языка BASIC программы *компилируются*. В результате онт выполняются значительно быстрее, и их переносимость улучшилась.

BASIC стал намного популярнее в 1991 году, когда компания Microsoft выпустила Visual Basic для Windows. Этот продукт облегчил массовую разработку самостоятельных приложений для Windows. Visual Basic имеет мало общего с ранними версиями BASIC, но последний представляет собой основу VBA.

Обзор VBA

Excel 5 — это первое приложение на рынке программного обеспечения, в котором появился Visual Basic for Applications (VBA). VBA считается стандартным языком написания сценариев для приложений Microsoft и в настоящее время входит в состав всех приложений Office 2010 и даже приложений других компаний. Следовательно, освоив VBA для Excel, вы сможете сразу перейти к созданию макросов для других программных продуктов Microsoft (равно как и приложений других компаний). Более того, вы сможете создавать полноценные программные продукты, одновременно использующие функции самых разных приложений.

Объектные модели

Секрет использования VBA заключается в правильном понимании *объектной модели* в каждом отдельном приложении. Следует отметить, что VBA всего лишь управляет объектами, а каждый программный продукт (Excel, Word, Access, PowerPoint и т.п.) имеет свою объектную модель. Приложением можно управлять программным образом только с помощью объектов, которые представлены в этом приложении.

Так, в объектной модели Excel представлено несколько мощных объектов анализа данных, например рабочие листы, диаграммы, сводные таблицы, сценарии, а также многочисленные математические, финансовые, инженерные и общие функции. С помощью VBA вы можете работать с этими объектами и разрабатывать автоматизированные процедуры. В процессе изучения VBA в Excel вы сможете получить полное представление об объектной модели. Сначала она покажется вам невероятно сложной. Однако со временем разрозненные фрагменты будут собраны в единую картину, и вы поймете, что овладели очень сложной темой!

Уйдет ли VBA в отставку

Последние несколько лет я только и слышу о том, что компания Microsoft собирается "убрать" VBA из приложений Office, заменив его платформой .NET. Эти слухи не имеют реальных оснований. Конечно, компания Microsoft использует другие методы автоматизации приложений Office, но VBA еще рано списывать со счетов, как и Excel для Windows. Впрочем, на момент написания этой книги компания Microsoft заявила, что VBA больше *не входит* в состав Excel для Macintosh. Невелика потеря, потому что большинство VBA-приложений Excel несовместимы с платформой Macintosh.

В чем залог успеха VBA? В миллионах основанных на этом языке проектах, а также в том, что VBA намного проще в изучении и применении, чем другие подобные языки.

Сравнение VBA и XLM

До появления Excel 5 разработчиками использовался мощный (но сложный для понимания) язык макросов под названием XLM. Более поздние версии Excel все еще выполняют макросы XLM, но начиная с Excel 97 пользователи не имеют возможности создавать макросы на языке XLM. Как разработчик вы должны знать о существовании XLM (на случай, если столкнетесь с макросами, написанными на этом языке), но для собственных разработок используйте исключительно VBA.

Примечание

Не путайте макроязык XLM с языком XML (eXtensible Markup Language — расширяемый язык разметки). Хотя в аббревиатурах, обозначающих эти языки, используются одни и те же буквы, между ними нет ничего общего. С помощью XML организуется формат, предназначенный для хранения структурированных данных. Формат XML по умолчанию применяется всеми приложениями Office 2010.

Основы VBA

Прежде чем непосредственно перейти к рассмотрению методов программирования, ознакомьтесь с материалом этого раздела, представляющим краткий обзор следующих разделов главы. Ниже указаны темы, которые будут рассмотрены в оставшихся разделах главы.

Далее кратко описана структура VBA.

- **Код.** Действия в VBA осуществляются в результате выполнения кода VBA.
- **Модуль.** Вы создаете (или записываете) программу VBA, которая сохраняется в модуле VBA. Модуль VBA состоит из процедур.
- **Процедуры.** Процедура, по существу, представляет собой элемент компьютерной программы, выполняющей определенное действие. В VBA поддерживаются два типа процедур: `Sub` (подпрограмма) и `Function` (функция).
  - `Sub`. Процедура `Sub` включает набор операторов, которые могут вызываться различными способами. Ниже приводится пример простой процедуры `Sub` под названием `Test`. Эта процедура вычисляет сумму, а затем отображает результат в окне сообщений.

```
Sub Test()
   Sum = 1 + 1
   MsgBox "Ответ " & Sum
End Sub
```

  - `Function`. Модуль VBA может также включать процедуры типа `Function`. Эта процедура возвращает единственное значение (иногда массив значений) и может вызываться из другой процедуры VBA или использоваться в формуле рабочего листа. Ниже приводится пример функции под названием `AddTwo`.

```
Function AddTwo(arg1, arg2)
   AddTwo = arg1 + arg2
End Function
```

- **Объекты.** VBA управляет объектами, которые представлены запускающим приложением (в данном случае — Excel). Excel позволяет управлять более чем ста классами объектов, включая рабочую книгу, рабочий лист, диапазон ячеек рабоче-

го листа, диаграмму и нарисованный прямоугольник. В вашем распоряжении находятся и другие объекты, с которыми можно работать в VBA. Классы объектов организованы в иерархическую структуру.

Объекты могут выступать контейнерами других объектов. Например, Excel — это объект под названием `Application`; он содержит другие объекты, например `Workbook` и `CommandBar`. Объект `Workbook` содержит другие объекты, например `Worksheet` и `Chart`. Объект `Worksheet` может включать такие объекты, как `Range`, `PivotTable` и т.д. Совокупность объектов называется *объектной моделью Excel*.

- **Коллекции.** Одинаковые объекты образуют *коллекцию*. Например, коллекция `Worksheets` включает все рабочие листы в указанной рабочей книге. Коллекция `CommandBars` включает все объекты `CommandBar`. Коллекции сами являются объектами.
- **Иерархия объектов.** При ссылке на объект, вложенный в другой объект, положение в иерархической структуре объектной модели задается с помощью точки-разделителя. Например, на рабочую книгу `Книга1.xlsx` можно сослаться так:

  ```
  Application.Workbooks("Книга1.xlsx")
  ```

  При этом устанавливается ссылка на рабочую книгу `Книга1.xlsx` в коллекции `Workbooks`. Эта коллекция содержится в объекте Excel под названием `Application`. Переходя на следующий уровень, можно сослаться на лист `Лист1` в книге `Книга1`:

  ```
  Application.Workbooks("Книга1.xlsx").Worksheets("Лист1")
  ```

  Можно также перейти на уровень ниже и сослаться на отдельную ячейку:

  ```
  Application.Workbooks("Книга1.xlsx").Worksheets("Лист1").Range("A1")
  ```

- **Активные объекты.** При опущенной ссылке на объект Excel использует активные объекты. Если активной рабочей книгой является `Книга1`, ссылка записывается в упрощенном виде:

  ```
  Worksheets("Лист1").Range("A1")
  ```

  Если известно, что активным листом является `Лист1`, ссылку можно упростить еще больше: `Range("A1")`.

- **Свойства объекта.** Объекты имеют свойства. Свойство может рассматриваться как настройка объекта. Например, объект диапазона имеет свойства `Value` и `Name`. Объекту диаграммы присущи свойства `HasTitle` и `Type`. Для определения (изменения) свойств объекта можно использовать VBA.

  Сослаться на свойство можно, записав объект и свойство, разделенные точкой. Например, сослаться на значение в ячейке A1 листа `Лист1` можно так:

  ```
  Worksheets("Лист1").Range("A1").Value
  ```

- **Переменные VBA.** Переменным VBA можно присваивать значения. Рассматривайте переменную как имя, которое применяется для хранения определенного значения. Так, если значение в ячейке A1 на листе `Лист1` нужно присвоить переменной `Interest`, используйте следующий оператор VBA:

  ```
  Interest = Worksheets("Лист1").Range("A1").Value
  ```

- **Методы объектов.** Все объекты имеют методы. *Метод* — это действие, которое выполняется над объектом. Например, один из методов объекта `Range` называется `ClearContents`. Он удаляет содержимое диапазона ячеек. Методы вводятся после названия объекта с методом, причем в качестве разделителя используется точка. Например, для удаления содержимого ячейки A1 с активного рабочего листа используется следующая конструкция: `Range("A1").ClearContents`.
- **Стандартные конструкции языков программирования.** В состав VBA входят все конструкции современных языков программирования, включая массивы, циклы и т.д.
- **События.** Некоторые объекты распознают определенные события, причем можно создать код VBA, который вызывается при наступлении определенных событий. Например, открытие рабочей книги приведет к вызову события `Workbook_Open`. Изменение ячейки в рабочей книге приведет к вызову события `Worksheet_Change`.

Следует отметить, что в этом разделе VBA описан достаточно полно. Теперь вам осталось изучить представленные возможности более подробно, чему и посвящена остальная часть главы.

Аналогия

Если вы любите аналогии, то эта врезка для вас. Возможно, она поможет вам лучше понять взаимосвязи между объектами, свойствами и методами в VBA. В этой аналогии Excel сравнивается с сетью ресторанов быстрого питания.

Основной объект Excel — `Workbook` (Рабочая книга). В сети ресторанов быстрого питания основным элементом является отдельный ресторан. В Excel можно добавлять либо закрывать рабочие книги, а набор открытых рабочих книг называется `Workbooks` (коллекция, состоящая из объектов `Workbook`). Аналогично, руководство сети ресторанов быстрого питания может открывать и закрывать рестораны, и все они в сети рассматриваются как коллекция `Restaurants` — набор из объектов `Restaurant`.

Рабочая книга Excel является объектом, но она содержит и другие объекты, такие как рабочие листы, диаграммы, модули VBA и т.д. Более того, каждый объект в рабочей книге может включать собственные объекты. Например, объект `Worksheet` может включать объекты `Range` (Диапазон), `PivotTable` (Сводная таблица), `Shape` (Фигура) и т.д.

По нашей аналогии ресторан быстрого питания (подобно рабочей книге) может включать такие объекты, как `Kitchen` (Кухня), `DiningArea` (Столовое помещение) и `Tables` (Столы), т.е. целую коллекцию. Более того, руководство может добавлять объекты в объект `Restaurant` или удалять их. Например, в коллекцию `Tables` (Столы) могут добавляться отдельные столы. Каждый из этих объектов может включать другие объекты. Например, объект `Kitchen` (Кухня) может включать такие объекты, как `Stove` (Плита), `VentilationFan` (Вытяжка), `Chef` (Шеф-повар), `Sink` (Раковина) и т.д.

Пока что все хорошо, и аналогия работает. Посмотрим, можно ли продолжить сравнение.

Объекты Excel имеют свойства. Например, объект `Range` (Диапазон) включает свойства `Value` (Значение) и `Name` (Имя), а объект `Shape` (Фигура) — `Width` (Ширина) и `Height` (Высота). Объекты из ресторана быстрого питания также имеют свойства. Например, объекту `Stove` (Плита) присущи такие свойства, как `Temperature` (Температура) и `NumberofBurners` (Количество конфорок). Объект `VentilationFan` (Вытяжка) имеет собственный набор свойств (`TurnedOn` (Включить), `RPM` (Изменить скорость вращения) и т.д.).

Помимо свойств, объекты Excel имеют методы, выполняющие операции над объектами. Например, метод `ClearContents` (Удаление содержимого) удаляет содержимое объекта

Range (Диапазон). Объект из ресторана быстрого питания также имеет свои методы. Так, у объекта Stove (Плита) нетрудно заметить метод ChangeThermostat (Изменить температуру), а объект VentilationFan (Вытяжка) включает метод SwitchOn (Включить).

В Excel методы иногда применяются для изменения свойств объекта. Например, метод ClearContents (Удаление содержимого), относящийся к объекту Range (Диапазон), применяется для изменения свойства Range Value (Значение диапазона). Аналогично метод ChangeThermostat (Изменить температуру) объекта Stove (Плита) применяется для изменения свойства Temperature (Температура).

Используя VBA, можно написать процедуры, позволяющие манипулировать объектами Excel. В ресторанах быстрого питания руководство может давать указания по работе с объектами ресторана. ("Включить плиту и переключить вытяжку на максимальную мощность.") Не правда ли, аналогия весьма прозрачная?

Знакомство с редактором Visual Basic

Все программы на языке VBA создаются с помощью редактора Visual Basic (Visual Basic Editor — VBE). Этот редактор представляет собой отдельное приложение, которое легкодоступно при работе с Excel. В данном случае под доступностью понимается то, что программа сама открывает редактор VBE, как только это понадобится пользователю. Единственное, чего нельзя сделать, — это запустить редактор VBE отдельно, поскольку сначала запускается на выполнение Excel.

Примечание

Модули VBA хранятся в файлах рабочих книг. Но эти модули нельзя увидеть до тех пор, пока не будет активизирована среда VBE.

Отображение вкладки Разработчик

На ленте Excel изначально вкладка Разработчик (Developer) не отображается. Без нее не обойтись, если вы собираетесь программировать на VBA.

1. Щелкните правой кнопкой мыши на ленте и в контекстном меню выберите параметр Настройка ленты (Customize the Ribbon).

   На экране появится раздел Настройка ленты (Customize Ribbon) диалогового окна Параметры Excel (Excel Options).

2. В отображенном в правой части окна списке установите флажок возле позиции Разработчик (Developer).
3. Щелкните на кнопке OK.

После выполнения перечисленных выше действий на ленте Excel появится новая вкладка (рис. 7.1).

Рис. 7.1. По умолчанию вкладка Разработчик не отображается

Запуск VBE

Во время работы в Excel вы можете перейти к окну VBE одним из следующих способов:

- нажать комбинацию клавиш <Alt+F11>;
- выбрать команду Разработчик⇨Код⇨Visual Basic (Developer⇨Code⇨Visual Basic).

Кроме того, можно получить доступ к двум специальным модулям. (Эти специальные VBA-модули используются процедурами обработчика событий, которые будут описаны в главе 19.)

- Щелкните правой кнопкой мыши на ярлыке листа, выберите пункт меню Исходный текст (View Code), и будет отображен модуль кода для рабочего листа.
- Щелкните правой кнопкой мыши на строке заголовка рабочей книги и выберите пункт Исходный текст (View Code). В результате выполнения этого действия отобразится модуль кода рабочей книги. Если окно рабочей книги максимизировано в Excel, строка заголовка не отображается.

Окно редактора VBE показано на рис. 7.2. Конечно, окно вашего редактора VBE может выглядеть иначе. Окно редактора VBE может легко изменяться — пользователь может скрывать окна, изменять их размеры, закреплять их, изменять порядок их расположения, а также выполнять некоторые другие операции.

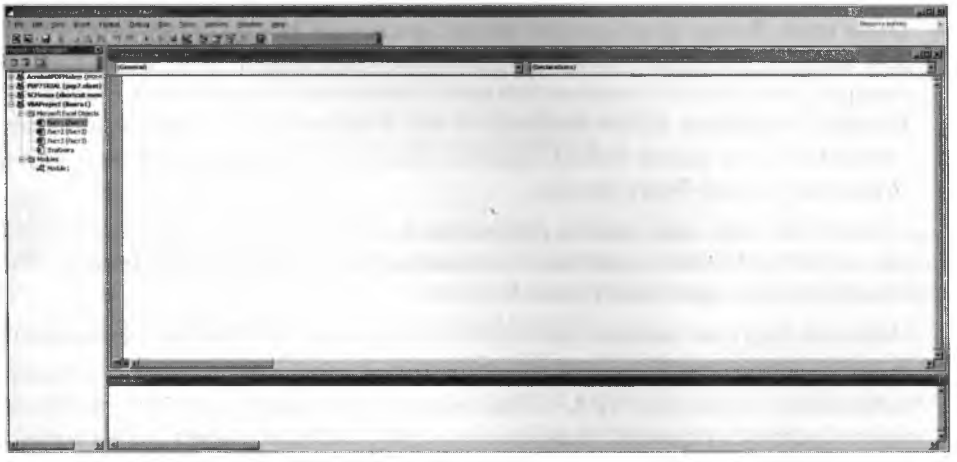


Рис. 7.2. Окно редактора Visual Basic

Окно VBE

Окно VBE включает ряд элементов. В следующих разделах кратко описаны ключевые компоненты окна редактора Visual Basic.

- **Строка меню.** Несмотря на повсеместное засилье в Excel ленточного интерфейса, редактор Visual Basic по-прежнему использует классические меню и панели инструментов. Строка меню VBE работает так же, как строка меню любого другого приложения. Она содержит команды, используемые для управления различными компонентами VBE. Кроме того, для выполнения многих команд меню используются комбинации клавиш. Например, для команды View⇨Immediate Window (Вид⇨Окно отладки) применяется комбинация клавиш <Ctrl+G>.

Совет

В VBE также представлены контекстные меню. Щелкнув правой кнопкой мыши практически на любом элементе окна VBE, вы увидите меню, предлагающее ряд команд.

- **Панели инструментов.** Стандартная панель инструментов Standard, которая по умолчанию находится под строкой меню. Это одна из шести панелей инструментов, используемых в VBE (строка меню также считается панелью инструментов). Панели инструментов VBE работают, как и в Excel: можно задавать специальные настройки для панелей инструментов, перемещать их, отображать другие панели инструментов и т.д. Для настройки панелей инструментов VBE используется команда View⇨Toolbars⇨Customize (Вид⇨Панели инструментов⇨Настройка).
- **Окно Project Explorer.** В окне Project Explorer отображается древовидная структура всех открытых в данный момент в Excel рабочих книг (включая надстройки и скрытые рабочие книги). Каждая рабочая книга известна как *проект*. Подробнее о Project Explorer будет сказано в следующем разделе.

  Если в редакторе Visual Basic окно Project Explorer не отображено, нажмите <Ctrl+R>. Чтобы скрыть его, щелкните на кнопке закрытия окна в строке заголовка (или щелкните правой кнопкой мыши в любом месте окна и выберите команду Hide (Скрыть) из контекстного меню) либо на кнопке Close (Закрыть) в строке заголовка.
- **Окно кода.** В окне кода (которое иногда называется Module) содержится код VBA. Для каждого элемента проекта представлено собственное окно кода. Чтобы просмотреть код объекта, дважды щелкните мышью на этом объекте в окне Project Explorer. Например, чтобы просмотреть код объекта `Лист1`, дважды щелкните на элементе `Лист1` в окне Project Explorer. Если вы не создавали для него VBA-код, открывшееся окно будет пустым.

  Существует еще один способ просмотра кода объекта — выделить этот объект в окне Project Explorer и щелкнуть на кнопке View Code (Просмотр кода) панели инструментов вверху окна Project Explorer.

  Окна кода будут рассмотрены далее в этой главе (см. раздел "Работа с окнами кода").
- **Окно отладки.** Окно отладки (Immediate) предназначено для непосредственного выполнения операторов VBA, тестирования операторов и отладки кода. Его можно отображать и скрывать. Если окна отладки в данный момент нет на экране, нажмите клавиши <Ctrl+G>. Чтобы закрыть окно отладки, щелкните на кнопке его закрытия в строке заголовка (или щелкните правой кнопкой мыши в любом месте окна и выберите опцию Hide (Скрыть) из контекстного меню).

Работа с Project Explorer

При работе в редакторе Visual Basic каждая рабочая книга Excel и открытые в данный момент надстройки рассматриваются как проекты. Проект можно считать коллекцией объектов, организованных в виде иерархической структуры. Вы раскроете проект, если щелкнете на знаке "плюс" слева от его названия в окне Project Explorer. Проект сворачивается после щелчка на знаке "минус" слева от его названия. При попытке развернуть проект, защищенный паролем, отображается окно для ввода пароля.

Примечание

В верхней части окна Project Explorer отображаются три пиктограммы. Находящаяся в правой части пиктограмма Toggle Folder (Переключить папку) определяет отображение объектов проекта в виде иерархического или простого списка.

На рис. 7.3 показано окно Project Explorer, в котором отображается несколько проектов.

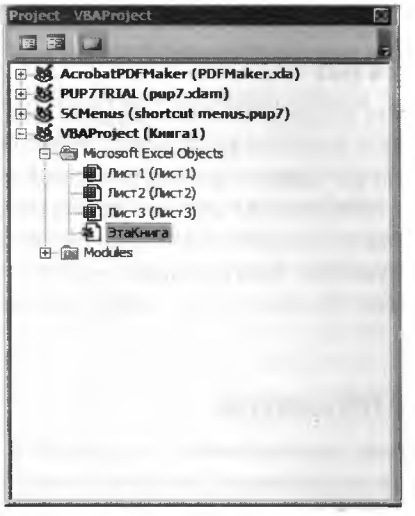


Рис. 7.3. В окне Project Explorer отображается несколько проектов

Предупреждение

При запуске VBE отображаемый программный модуль не всегда соответствует объекту, который выделен в окне Project Explorer. Чтобы убедиться в том, что работа происходит с нужным программным модулем, дважды щелкните на объекте в окне Project Explorer.

Если в Excel загружено несколько рабочих книг и надстроек, окно Project Explorer будет загромождено. К сожалению, скрыть проекты в окне Project Explorer невозможно. Но если вы детально не рассматриваете отдельные проекты, то можете отображать их в свернутом виде.

Развернутое дерево каждого проекта имеет минимум один узел под названием Microsoft Excel Objects. В нем содержатся элементы каждого рабочего листа и лист диаграмм рабочей книги (рабочий лист считается объектом), а также объект `ЭтаКнига` (ThisWorkbook), представляющий объект `ActiveWorkbook`. Если в проекте используются модули VBA, то в дереве отображается также узел `Modules`, в котором перечислены модули. Проект может включать узел `Forms`, содержащий объекты `UserForm` (пользовательские формы, известные как пользовательские диалоговые окна). Если в проекте находятся модули классов, то в дереве отображается узел `Class Modules`. Аналогично, если проект включает ссылки, отображается узел `References` (хотя на самом деле польза от этого узла сомнительна, поскольку ссылки не могут включать VBA-код).

Добавление нового модуля VBA

Чтобы добавить в проект новый модуль VBA, выделите название проекта в окне Project Explorer и выберите команду Insert⇨Module (Вставка⇨Модуль). Также можете щелкнуть правой кнопкой мыши на названии проекта и выбрать команду Insert⇨Module в контекстном меню.

При создании макроса Excel автоматически включает модуль VBA, в котором будет храниться создаваемый вами код.

Удаление модуля VBA

Чтобы удалить из проекта модуль VBA или модуль класса, выделите название модуля в окне Project Explorer и используйте команду File⇨Remove *xxx* (где *xxx* — название модуля). Кроме того, вы можете щелкнуть правой кнопкой мыши на названии модуля и выбрать команду Remove *xxx* из контекстного меню. Перед удалением модуля отображается запрос на его экспорт (детально об этом рассказано в следующем разделе). Вы не сможете удалить программные модули, соответствующие рабочей книге (программный модуль `ЭтаКнига`), а также рабочему листу (например, программный модуль `Лист1`).

Экспорт и импорт объектов

За исключением объектов, перечисленных в узле `References`, каждый объект в проекте можно сохранить в отдельном файле. Сохранение отдельного объекта в проекте называется экспортом. Соответственно, вы можете также импортировать объекты в проект. Экспорт и импорт объектов востребован, если созданный ранее объект (например, модуль VBA или форму `UserForm`) нужно использовать в другом проекте.

Чтобы экспортировать объект, выберите его в окне Project Explorer и выполните команду File⇨Export File (Файл⇨Экспорт файла) (или нажмите <Ctrl+E>). При этом отображается диалоговое окно, запрашивающее имя файла. Обратите внимание, что сам объект остается в проекте (экспортируется только его копия). Если вы экспортируете объект `UserForm`, экспортируется также весь код, связанный с формой `UserForm`.

Чтобы импортировать файл в проект, выберите имя проекта в окне Project Explorer и выполните команду File⇨Import File (Файл⇨Импорт файла). Появится диалоговое окно, в котором необходимо указать имя файла. Можно импортировать только те файлы, которые экспортированы с помощью команды File⇨Export File.

Совет

Если нужно скопировать модуль или объект `UserForm` в другой объект, вовсе не обязательно сначала экспортировать, а затем импортировать объект. Убедитесь в том, что оба проекта открыты; просто откройте Project Explorer и перетащите объект из одного проекта в другой.

Работа с окнами кода

В совершенстве овладев VBA, вы будете много времени работать в окнах кода. Каждому объекту в проекте соответствует свое окно кода. Такими объектами могут быть следующие:

- сама рабочая книга (`ЭтаКнига` в окне Project Explorer);
- рабочий лист или лист диаграммы рабочей книги (например, `Лист1` или `Диаграмма1` в окне Project Explorer);
- модуль VBA;
- модуль класса (специальный тип модуля, позволяющий создавать новые классы объектов);
- форма `UserForm`.

Сворачивание и восстановление окон

В любой момент в редакторе VBE можно открыть несколько окон кода, и это существенно усложняет работу. Окна кода во многом напоминают окна рабочих листов Excel. Их можно свернуть, развернуть, скрыть, изменить порядок отображения на экране и т.д. Многие пользователи предпочитают разворачивать окно кода, над которым они работают в данный момент, что позволяет видеть большую часть программы, не отвлекаясь на другие модули. Чтобы максимизировать окно кода, щелкните на кнопке **Развернуть** в строке заголовка или дважды щелкните на самой строке заголовка. Чтобы вернуть окну кода прежний размер, щелкните на кнопке **Восстановить** в строке заголовка.

Иногда необходимо, чтобы на экране отображалось сразу несколько окон кода. Например, требуется сравнить код в двух модулях или скопировать код из одного модуля в другой.

Сворачивая окно кода, вы скрываете его в окне редактора. Кроме того, можно щелкнуть на кнопке **Свернуть** (Close) в строке заголовка окна кода, чтобы полностью его закрыть. Открыть окно кода заново можно, дважды щелкнув на соответствующем объекте в окне Project Explorer.

VBE не позволяет закрывать рабочую книгу. Для этого вы должны вернуться в окно Excel и там закрыть книгу. Однако можно использовать окно отладки (Immediate), чтобы закрыть рабочую книгу или отключить надстройку. Активизируйте это окно, введите оператор VBA (пример показан ниже) и нажмите <Enter>.

```
Workbooks("myaddin.xlam").Close
```

Этот оператор выполняет метод `Close` объекта `Workbook`, закрывающий рабочую книгу. В данном примере рабочая книга является надстройкой.

Сохранение кода VBA

Как правило, окно кода содержит четыре типа кода.

- **`Sub` (Процедура).** Это набор инструкций, выполняющих определенное действие.
- **`Function` (Функция).** Это набор инструкций, возвращающий значение или массив значений (функции VBA напоминают функции рабочего листа Excel, например `СУММ`).
- **`Property` (Процедуры свойств).** Специальные процедуры, используемые в модулях классов.
- **Объявление.** Объявления включают информацию о переменной, которая предоставляется VBA. Например, можно объявить тип данных для переменных, которые вы планируете использовать в коде.

В отдельном модуле VBA может храниться любое количество процедур, функций и объявлений. Способ организации модуля VBA зависит только от вашего желания. Можно записывать весь код VBA приложения в одном модуле VBA или разделять код на несколько модулей.

Примечание

Несмотря на то что пользователям предоставляются широкие возможности по выбору места хранения кода VBA, существует ряд ограничений. Процедуры обработки событий должны содержаться в окне кода объекта, которому соответствует это событие. Например, если создана процедура, которая вызывается при открытии рабочей книги, то эта процедура должна располагаться в окне кода для объекта `ЭтаКнига` и иметь специальное название. Подобный вопрос станет более понятным после того, как вы изучите события (глава 19) и пользовательские формы (часть IV).

Ввод кода VBA

Для того чтобы выполнить одно из действий программным образом, необходимо написать программу VBA в окне кода. Код VBA располагается в процедуре. Процедура состоит из операторов VBA. На данном этапе (для примера) остановимся только на одном типе окна кода — модуле VBA.

Вы можете добавить код в модуль VBA тремя способами.

- **Ввести код вручную.** Для этого используйте клавиатуру.
- **Использовать функцию создания макросов.** Используйте функцию записи макросов в Excel, чтобы записать действия и преобразовать их в код VBA.
- **Использовать операцию копирования и вставки.** Скопируйте текст программы из другого модуля и вставьте его в модуль, над которым работаете.

Ввод кода вручную

Иногда самый простой путь является наилучшим. Непосредственное введение кода связано с использованием клавиатуры, т.е. вы вводите код программы с помощью клавиатуры. Клавиша <Tab> при этом поможет задать отступ в строках, которые логически принадлежат одной группе (например, условные операторы `If` и `End If`). Это совершенно не обязательно, но помогает быстрее освоить программу, анализируя ее блочную структуру. Именно поэтому подобный подход в программировании называется "хорошим стилем".

Ввод и редактирование кода в модуле VBA выполняются обычным образом. Вы можете выделять текст, копировать и вырезать, а затем вставлять в другое место программы.

Терминология

В книге применяются термины *подпрограмма*, *процедура* и *макрос*. Программисты для описания автоматизированной задачи обычно используют слово *процедура*. В Excel процедуру также называют *макросом*. Технически процедура может быть двух видов: `Sub` (Подпрограмма) либо `Function` (Функция); оба вида иногда называют *подпрограммами*. В книге эти термины используются как синонимы. Тем не менее между процедурами типа `Sub` и `Function` существует большая разница, о чем будет рассказано в главах 9 и 10.

Отдельная инструкция в VBA может иметь произвольную длину. Однако для обеспечения удобочитаемости кода длинные инструкции лучше разбить на две или более строк. Для этого следует в конце строки ввести пробел и символ подчеркивания, а затем нажать <Enter> и продолжить инструкцию в следующей строке. Например, ниже приведен один оператор VBA, разбитый на четыре строки.

```
MsgBox "Невозможно найти " & UCase(SHORTCUTMENUFILE) _
   & vbCrLf & vbCrLf & "Файл должен находиться в " _
   & ThisWorkbook.Path & vbCrLf & vbCrLf _
   & "Возможно, требуется переустановить BudgetMan", _
   vbCritical, APPNAME
```

Обратите внимание, что четыре последние строки этого оператора введены с отступом. Это необязательное условие, однако таким образом вы указываете, что на самом деле эти четыре строки являются одним оператором.

Совет

Как и в Excel, в VBE существует несколько уровней отмены операций. Поэтому, если вы по ошибке удалили инструкцию, можете несколько раз щелкнуть на кнопке Undo (Отменить) (либо нажать клавиши <Ctrl+Z>), после чего инструкция вновь появится в коде. После отмены операции можно щелкнуть на кнопке Redo (Вернуть) (либо нажать клавиши <Ctrl+Y>), чтобы вернуть изменения, которые ранее были отменены. Эта функция поможет исправить критически важные ошибки, поэтому не пренебрегайте ею.

Выполните такие действия: добавьте в проект модуль VBA и введите следующую процедуру в окне кода данного модуля.

```
Sub SayHello()
   Msg = "Ваше имя " & Application.UserName & "?"
   Ans = MsgBox(Msg, vbYesNo)
   If Ans = vbNo Then
      MsgBox "Ничего страшного."
   Else
      MsgBox "Наверное, я ясновидящий!"
   End If
End Sub
```

На рис. 7.4 показано, как это выглядит в модуле VBA.

Примечание

При вводе кода вы могли заметить, что VBE вносит некоторые изменения во введенный текст. Например, если пропустить пробел перед или после знака равенства (=), VBE автоматически вставит его. Кроме того, изменяется цвет некоторых слов кода. Это нормально, и позже вы оцените данный факт.

Для вызова на выполнение процедуры `SayHello` убедитесь в том, что текстовый курсор находится в области вводимого текста. Затем выполните следующие действия.

1. Нажмите клавишу <F5>.
2. Выберите команду Run⇨Run Sub/UserForm (Выполнить⇨Выполнить процедуру/пользовательскую форму).
3. Щелкните на кнопке Run Sub/UserForm, находящейся на панели инструментов Standard.

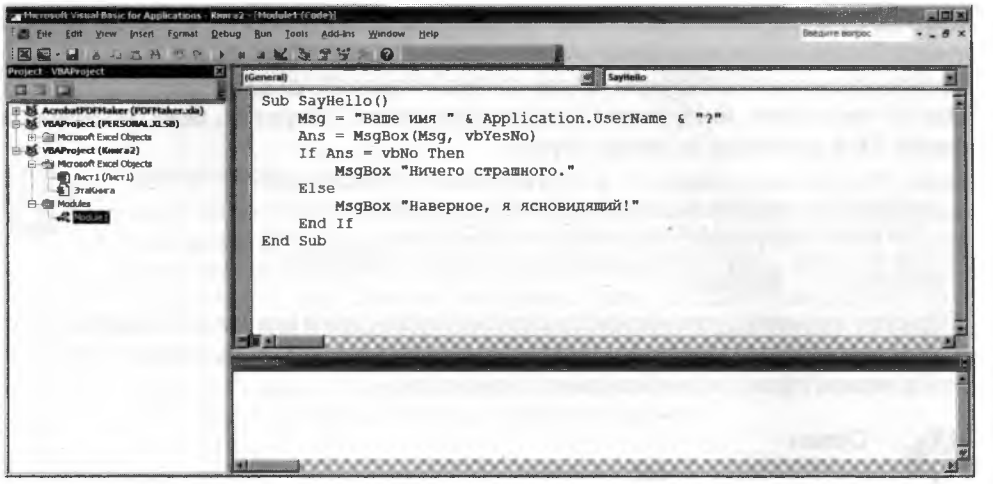


Рис. 7.4. Ваша первая VBA-процедура

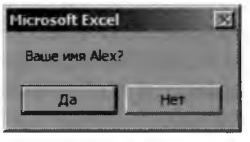


Рис. 7.5. Результат выполнения процедуры, код которой показан на рис. 7.4

Если вы ввели код правильно, процедура будет выполнена. Вы сможете выбрать ответ в простом диалоговом окне (рис. 7.5), в котором отображается имя пользователя, заданное в диалоговом окне Параметры Excel (Excel Options). Обратите внимание, что при выполнении макроса активизируется программа Excel. На данном этапе вам не обязательно понимать принципы работы программы; в этом вы разберетесь, прочитав следующие разделы и главы книги.

Примечание

В большинстве случаев вы будете запускать макросы в Excel. Однако тестировать макрос удобнее, выполняя его непосредственно в VBE.

Написанная программа представляет собой процедуру VBA (называемую также *макросом*). Когда вы даете команду выполнить макрос, VBE быстро компилирует код и запускает его. Другими словами, каждая инструкция анализируется в VBE, а Excel всего лишь выполняет то, что указано в инструкциях. Вы можете выполнить макрос сколько угодно раз, хотя через некоторое время он теряет свою привлекательность.

При выполнении кода этой простой процедуры выполнялись следующие действия (все они подробно рассмотрены далее):

- объявление процедуры (первая строка);
- присвоение значения переменным (`Msg` и `Ans`);
- конкатенация строк (с помощью оператора `&`);
- использование встроенной функции VBA (`MsgBox`);
- применение встроенных констант VBA (`vbYesNo` и `vbNo`);
- использование конструкции `If-Then-Else`;
- окончание процедуры (последняя строка).

Неплохо для первого раза, не правда ли?

Использование средства записи макросов

Одним из способов создания кода модуля VBA является запись последовательности действий с помощью специальной функции записи макросов Excel.

Как бы вы ни старались, но записать показанную в предыдущем примере процедуру `SayHello` вы не сможете. Запись макросов — полезное средство, но оно имеет свои ограничения. После записи макросов вы неоднократно будете вносить изменения или вводить дополнительный код вручную.

В следующем примере показано, как записать макрос, изменяющий ориентацию страницы на альбомную. Если вы хотите получить его самостоятельно, то начните работу с пустой рабочей книги и выполните следующие действия.

1. Активизируйте рабочий лист в книге (подойдет любой лист).
2. Выберите команду **Разработчик⇨Код⇨Запись макроса** (Developer⇨Code⇨Record Macro). После этого Excel отобразит диалоговое окно **Запись макроса** (Record Macro).
3. Щелкните на кнопке **OK**, чтобы принять параметры, заданные по умолчанию.

   Excel автоматически вставит новый модуль VBA в проект VBA рабочей книги. Начиная с этого момента Excel будет преобразовывать все ваши действия в код VBA. Обратите внимание, что в строке состояния Excel отображается голубой квадратик. После щелчка на нем запись макроса прекращается.
4. Выберите команду **Разметка страницы⇨Параметры страницы⇨Ориентация⇨Альбомная** (Page Layout⇨Page Setup⇨Orientation⇨Landscape).
5. Выберите команду **Разрабочик⇨Код⇨Остановить запись** (Developer⇨Code⇨Stop Recording) или щелкните на голубом квадратике, который находится в строке состояния.

   Программа Excel прекратит запись ваших действий.

Чтобы просмотреть макрос, запустите VBE (проще всего нажать клавиши <Alt+F11>) и найдите проект в окне Project Explorer. Щелкните на узле `Modules`, чтобы развернуть его. Затем щелкните на элементе `Module1`, чтобы отобразить окно кода (если в проекте уже присутствовал модуль `Module1`, новый макрос будет находиться в модуле `Module2`). Код, созданный всего лишь одной простой командой Excel, представлен на рис. 7.6. Строки кода с предшествующими им апострофами являются комментариями и не выполняются.

Возможно, вас удивит объем кода, сгенерированного всего лишь одной командой (особенно если вы записываете макрос впервые). Несмотря на то что вы изменили только одну простую настройку на вкладке **Параметры страницы** (Page Setup), Excel генерирует код, задающий все параметры в этом диалоговом окне.

Таким образом, зачастую программа, полученная при записи макроса, избыточна. Если вы хотите, чтобы макрос всего лишь изменял ориентацию страницы на альбомную, можете значительно упростить его, удалив ненужный код. Это облегчит восприятие макроса и ускорит его выполнение, поскольку избавит его от лишних операций. Упростить макрос вы вправе до следующего вида.

```
Sub Macro1()
   With ActiveSheet.PageSetup
      .Orientation = xlLandscape
   End With
End Sub
```

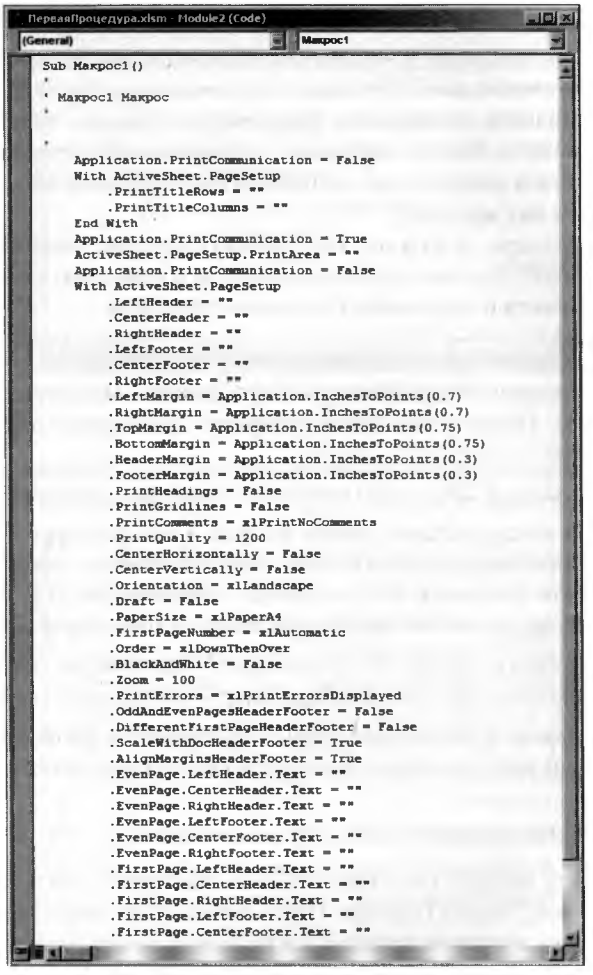


Рис. 7.6. Код, сгенерированный функцией записи макросов Excel

Мы удалили весь код, кроме строки, изменяющей свойство `Orientation`. На самом деле данный макрос можно упростить еще больше, так как конструкция `With-End With` не обязательна при изменении только одного свойства.

```
Sub Макрос1()
   ActiveSheet.PageSetup.Orientation = xlLandscape
End Sub
```

В данном примере макрос изменяет свойство `Orientation` объекта `PageSetup` активного листа. Отметим, что `xlLandscape` — это заранее заданная константа, которая предназначена для изменения ориентации страницы. Переменная `xlLandscape` имеет значение **2**, а `xlPortrait` — значение **1**. Следующий макрос работает, как и предыдущий `Макрос1`.

```
Sub Макрос1а()
   ActiveSheet.PageSetup.Orientation = 2
End Sub
```

Многим пользователям легче запомнить название константы, чем произвольные числа. Вы можете воспользоваться справочной системой, чтобы выучить соответствующие каждой настройке константы.

Зачастую данная процедура вводится непосредственно в модуль VBA, но для этого необходимо знать, какие объекты, свойства и методы требуется использовать. Но ведь записать макрос быстрее. Кроме того, данный пример продемонстрировал наличие у объекта `PageSetup` свойства `Orientation`.

Примечание

Запись действий — это *наилучший* способ изучить VBA. Если у вас возникают проблемы с введением кода, воспользуйтесь функцией записи действий. Даже если вы получаете совсем не то, чего ожидали, результирующий код укажет правильное направление. Для получения информации об объектах, свойствах и методах, которые присутствуют в записанном коде, используйте электронную справочную систему.

Перекрестная ссылка

Механизм записи макросов подробнее рассматривается в разделе "Средство записи макросов".

Копирование кода VBA

Выше были рассмотрены способы непосредственного ввода кода и записи действий для создания программы VBA. Последний метод добавления кода в модуль VBA — копирование текста программы из другого модуля. Например, вы могли написать процедуру в одном из более ранних проектов, которая также используется в текущем проекте. Вместо того чтобы заново вводить код, достаточно открыть рабочую книгу, активизировать модуль и использовать стандартные способы копирования и вставки, чтобы скопировать его в текущий модуль VBA. Если после вставки возникает необходимость, подкорректируйте код модуля.

И не забывайте об Интернете. На веб-сайтах, форумах и в блогах можно найти тысячи примеров кода VBA. Достаточно скопировать нужный пример в окне браузера и вставить его в окно модуля VBA.

Совет

Как уже отмечалось в этой главе, можно импортировать в файл модуль, который был ранее экспортирован.

Настройка среды VBE

В процессе программирования в Excel вы будете проводить много времени, работая в окнах VBE. Чтобы сделать редактор более удобным, вы можете настроить некоторые его параметры.

В строке меню окна VBE выберите команду Tools⇨Options (Сервис⇨Параметры). Появится диалоговое окно Options (Параметры) с четырьмя вкладками: Editor (Редактор), Editor Format (Формат редактора), General (Общие) и Docking (Прикрепление). Некоторые наиболее часто используемые параметры этих вкладок будут рассмотрены в следующих разделах. Кстати, не путайте это окно с диалоговым окном Параметры

Excel (Excel Options) программы Excel, которое можно открыть в Excel с помощью команды Файл⇨Параметры Excel (File⇨Excel Options).

Вкладка Editor

На рис. 7.7 показаны параметры, доступ к которым можно получить, щелкнув на вкладке Editor (Редактор) диалогового окна Options (Параметры).

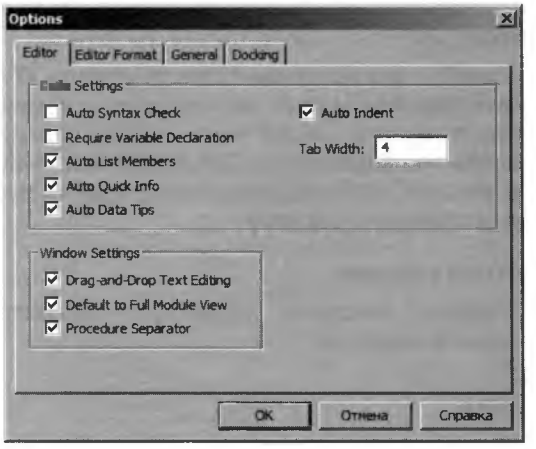


Рис. 7.7. Вкладка Editor диалогового окна Options

Параметр Auto Syntax Check

Настройка Auto Syntax Check (Автоматическая проверка синтаксиса) определяет, будет ли появляться диалоговое окно, когда VBE обнаружит синтаксическую ошибку в коде VBA. В этом диалоговом окне указывается тип допущенной ошибки. Если отменить установку этого флажка, то VBE выделит синтаксические ошибки, отобразив соответствующие фрагменты кода другим цветом, и вам не придется работать в диалоговых окнах, которые появляются на экране.

Я обычно отключаю параметр Auto Syntax Check, поскольку без труда могу определить, в чем заключается ошибка. Но если вы только начинаете работать с VBA, то дополнительная помощь вам не помешает.

Параметр Require Variable Declaration

При установленном параметре Require Variable Declaration (Обязательное декларирование переменных) VBE вставляет в начале каждого нового модуля следующий оператор:

```
Option Explicit
```

Если в модуле задан этот оператор, то вы должны явно определить каждую используемую в нем переменную. Таким образом, у вас вырабатывается хорошая привычка, которая, правда, требует дополнительных усилий. Если вы не объявляете переменные, все они имеют тип данных `Variant`; это достаточно гибко, но неэффективно с точки зрения использования аппаратных ресурсов и скорости выполнения кода. Более подробно объявление типа переменных рассматривается в главе 8.

Примечание

Изменение параметра Require Variable Declaration влияет на новые модули, а не на существующие.

Параметр Auto List Members

Если выбрана опция Auto List Members (Автоматическая вставка объектов), VBE предоставляет помощь при вводе кода VBA, отображая список элементов текущего объекта. К этим элементам относятся методы и свойства объекта, название которого вводится вручную.

Данный параметр весьма полезен, поэтому его рекомендуется всегда активизировать. На рис. 7.8 показан пример использования параметра Auto List Members (предназначение которого станет понятнее, когда вы начнете вводить код VBA самостоятельно). В данном примере VBE отображает список элементов объекта `Application`. Можете выбрать элемент из списка, чтобы не вводить его с помощью клавиатуры (в результате название элемента будет введено без ошибок).

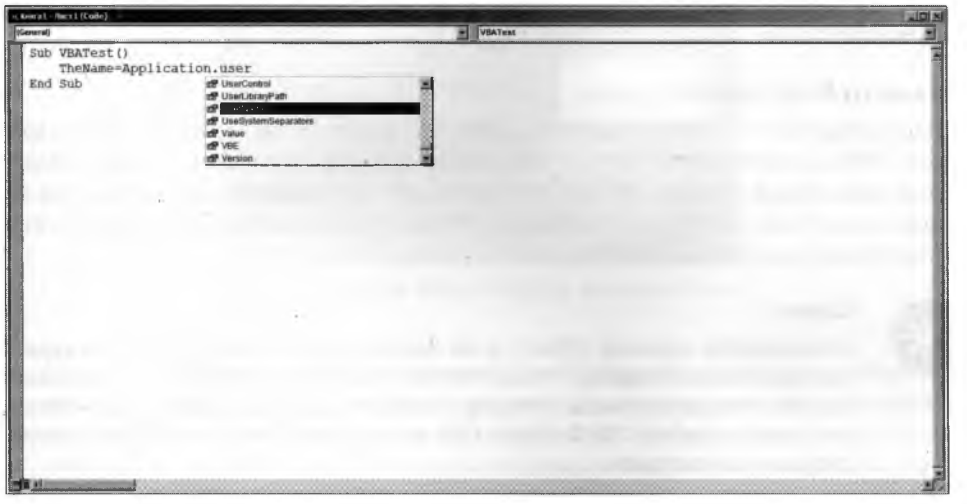


Рис. 7.8. Пример использования параметра Auto List Members

Параметр Auto Quick Info

Если включен параметр Auto Quick Info (Отображать краткие сведения), VBE будет отображать информацию об аргументах функций, свойств и методов, названия которых вы вводите с клавиатуры. Это очень полезно, поэтому рекомендуется всегда оставлять эту настройку включенной. На рис. 7.9 данная функция показана в действии: отображается синтаксис свойства `Cells`.

Параметр Auto Data Tips

Если включен параметр Auto Data Tips, VBE отображает при отладке кода значение переменной, над которой находится указатель мыши. Ознакомившись с инструментами отладки в VBE, вы сможете по достоинству оценить этот параметр. Рекомендуется всегда держать его включенным.

Рис. 7.9. Пример использования функции Auto Quick Info, предоставляющей сведения о свойстве `Cells`

Параметр Auto Indent

Настройка Auto Indent (Автоматический отступ) определяет, располагает ли автоматически VBE каждую новую строку программы с тем же отступом, который задан для предыдущей строки. Тем, кто использует отступы в программных кодах, советуем всегда обращаться к этому параметру. Вы можете также задать количество символов в отступе (по умолчанию указано значение 4).

Совет

Используйте клавишу <Tab>, а не пробел, чтобы задать отступ в коде. При использовании клавиши <Tab> отступы получаются более "правильными". Кроме того, для отмены отступа в конкретной строке воспользуйтесь комбинацией клавиш <Shift+Tab>. Она может применяться при выделении нескольких операторов.

Параметр Drag-And-Drop Text Editing

При выборе параметра Drag-and-Drop Text Editing (Включить редактирование перетаскиванием) вы можете копировать и перемещать текст, перетаскивая его с помощью мыши. Например, я оставляю этот параметр включенным, но никогда не пользуюсь функцией перетаскивания, так как предпочитаю для копирования и вставки обращаться к комбинациям клавиш.

Параметр Default To Full Module View

Параметр Default to Full Module View (По умолчанию использовать полный режим просмотра) определяет принцип просмотра процедуры. Если он включен, процедуры в окне кода помещаются в одно окно с полосой прокрутки. Если же он отключен, то вы можете просмотреть в определенный момент только одну процедуру. Рекомендуем активизировать этот параметр.

Параметр Procedure Separator

Когда параметр Procedure Separator (Разделение процедур) включен, в конце каждой процедуры в окне кода отображаются специальные разделители. Если вам нравятся эти визуальные подсказки окончания процедуры, выставляйте данный флажок.

Вкладка Editor Format

На рис. 7.10 показана вкладка Editor Format (Формат редактора), которая находится в диалоговом окне Options (Параметры). Параметры на этой вкладке определяют порядок отображения VBE.

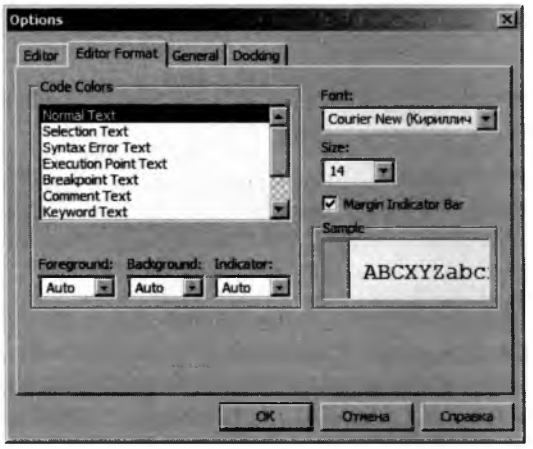


Рис. 7.10. Вкладка Editor Format в диалоговом окне Options

- **Параметр Code Colors (Цвета кода).** Предоставляет возможность выбрать цвета кода (текста и фона) и индикатора, который используется для выделения разных элементов программы VBA. Цвета, конечно, выбираются в зависимости от личных предпочтений. Вы можете согласиться с цветами, принятыми по умолчанию. Однако для разнообразия можете изменить эти настройки.
- **Параметр Font (Шрифт).** Предоставляет возможность указать шрифт, используемый в модулях VBA. Наибольшая эффективность достигается при работе с моноширинным шрифтом (например, Courier New). В таком шрифте все символы имеют одинаковую ширину, что делает программу более удобной для восприятия и анализа, так как все символы одинаково выровнены; кроме того, хорошо видны пробелы между словами.
- **Список Size (Размер).** Определяет размер шрифта кода модулей VBA. Эта настройка зависит от личных предпочтений, которые, в свою очередь, определяются разрешением монитора и вашим зрением. По умолчанию размер задан равным 10.
- **Параметр Margin Indicator Bar (Полоса индикатора границы).** Отображает вертикальную полосу вдоль левой границы окна кода, на которой высвечиваются всевозможные индикаторы. Его необходимо выставить; в противном случае вы не увидите полезные графические извещения при отладке кода.

Вкладка General

На рис. 7.11 показаны параметры, доступные на вкладке General (Общие) диалогового окна Options (Параметры).

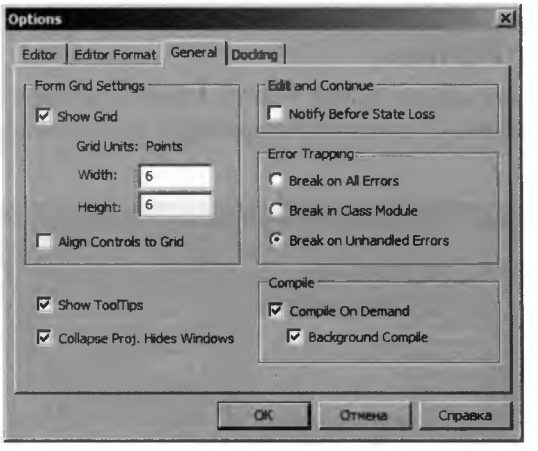


Рис. 7.11. Вкладка General, выбранная в диалоговом окне Options

- **Раздел Form Grid Settings (Параметры сетки формы).** Находящиеся в этом разделе параметры предназначены для настройки пользовательских диалоговых окон; с их помощью определяются параметры сетки, которая облегчает позиционирование элементов управления в окне `UserForm`. Если вы имеете некоторый опыт работы с пользовательскими диалоговыми окнами, то легко определите параметры сетки, которые будут полезны в работе.
- **Флажок Show ToolTips (Показать подсказки).** Определяет поведение кнопок панели инструментов. Рекомендуется всегда его устанавливать.
- **Флажок Collapse Proj. Hides Windows (Скрывать окна при сворачивании проектов).** Установка этого флажка приведет к автоматическому закрытию окон в случае сворачивания проекта в окне Project. Рекомендуется оставлять этот флажок установленным.
- **Раздел Edit and Continue (Редактировать и продолжать).** Содержит единственный флажок, который может оказаться полезным в процессе отладки. В случае установки этого флажка VBA отображает сообщение, если переменные теряют присвоенные им значения в результате появления каких-либо проблем.
- **Раздел Error Trapping (Перехват ошибок).** Находящиеся в этом разделе параметры определяют, что происходит при возникновении ошибки. Если вы создаете процедуры обработки ошибок, убедитесь в том, что установлен переключатель Break on Unhandled Errors (Остановка при возникновении неисправимой ошибки). При заданном параметре Break on All Errors (Остановка при возникновении любой ошибки) процедуры обработки ошибок игнорируются (вряд ли это нужно). Методы обработки ошибок подробно описываются в главе 9.
- **Раздел Compile (Компиляция).** Находящиеся в этом разделе два параметра управляют процессом компиляции кода. Рекомендуется установить оба этих па-

раметра. На современных компьютерах компиляция кода происходит практически мгновенно, если его размеры сравнительно невелики.

Вкладка Docking

На рис. 7.12 показана вкладка Docking (Прикрепление) диалогового окна Options (Параметры). Ее параметры определяют поведение нескольких окон редактора VBE — отображаются окна, которые могут быть прикреплены. Когда окно прикреплено, оно фиксируется по отношению к одной из границ окна VBE. В результате намного легче найти вспомогательное окно, так как оно отображается в строго определенной области. Если вы отключите все параметры прикрепления, то окна перемешаются между собой, а это усложнит работу. Как правило, идеальным выбором будут настройки по умолчанию.

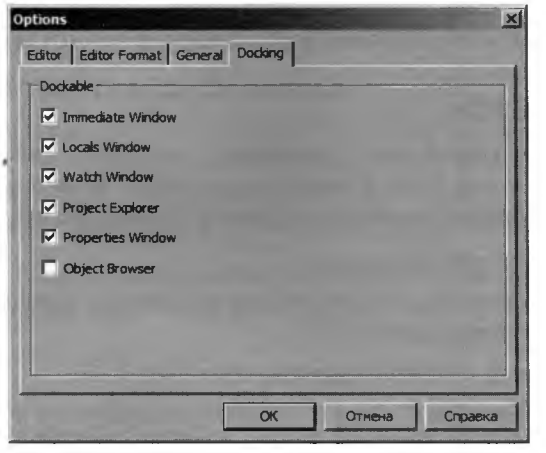


Рис. 7.12. Вкладка Docking диалогового окна Options

Для прикрепления окна просто перетащите его в новое место. Например, вам может понадобиться присоединить окно Project Explorer к левой границе окна. Захватите его за заголовок и переместите влево. Отпустите кнопку мыши в момент, когда окно "прилипнет" к левому краю экрана.

Примечание

Прикрепление окна в VBE всегда было основной проблемой при настройке рабочей среды, однако после многочисленных попыток вам, будем надеяться, все же удастся решить эту задачу.

Средство записи макросов

Ранее уже описывалось средство записи макросов, которое преобразует ваши действия в Excel в код на языке VBA. Рассмотрим это средство подробнее.

Примечание

А теперь самое время еще раз проверить, отображается ли на ленте Excel вкладка Разработчик (Developer). Если этой вкладки не видно, обратитесь за инструкциями к разделу "Отображение вкладки Разработчик".

Средство записи макросов — это чрезвычайно полезный инструмент, но не забывайте о следующих моментах.

- Запись лучше всего выполнять для простых макросов или небольшого фрагмента более сложного макроса.
- Не все выполняемые в Excel действия могут записываться.
- Команда записи макросов не может генерировать код, который включает циклические структуры (т.е. повторяющиеся операторы), а также присваивает переменные, выполняет условные операторы, отображает диалоговые окна и т.д.
- Средство записи макросов всегда создает процедуры типа `Sub`. С ее помощью невозможно создать процедуру типа `Function`.
- Созданная в результате записи программа зависит от определенных вами настроек.
- Вам придется часто дорабатывать записанный код, чтобы удалить лишние команды.

Что записывается

Средство записи макросов Excel преобразует действия, выполненные с помощью мыши и клавиатуры, в код VBA. Принцип выполнения можно описать на нескольких страницах, но целесообразнее рассмотреть пример. Выполните следующие действия.

1. Начните с пустой рабочей книги.
2. Убедитесь, что окно Excel полностью не развернуто. Добейтесь, чтобы на экране оставалось свободное место.
3. Нажмите клавиши <Alt+F11> для открытия окна VBE.

   Примечание. Убедитесь в том, что это окно развернуто не полностью. Если это условие не выполняется, одновременный просмотр окон VBE и Excel невозможен.
4. Измените размер и разместите окна Excel и VBE таким образом, чтобы возможен был их просмотр. (Для достижения наилучших результатов сверните окна всех выполняющихся приложений.)
5. Перейдите в окно Excel, выберите команду **Разработчик⇨Код⇨Запись макроса** (Developer⇨Code⇨Record Macro) и щелкните на кнопке **OK** для запуска функции записи макросов.
6. Перейдите в окно VBE.
7. В окне Project Explorer дважды щелкните на узле `Module1` для отображения модуля в окне кода.
8. Закройте окно Project Explorer в среде VBE для лучшего просмотра окна кода.

В результате получим экран, показанный на рис. 7.13. Размер окон зависит от выбранного разрешения экрана. Если вы являетесь счастливым обладателем двухмониторной системы, на первом мониторе отобразите окно VBA, а на втором — окно Excel.

Теперь поработайте на рабочем листе, выбирая разные команды Excel. Посмотрите, как генерируется код в окне, представляющем модуль VBA. Вам следует выполнить несколько действий: выделить ячейки, ввести данные, изменить формат ячеек. Далее используйте команды ленты, создайте диаграмму, поработайте с графическими объектами и т.д. Все прояснится, когда на экране появится код программы.

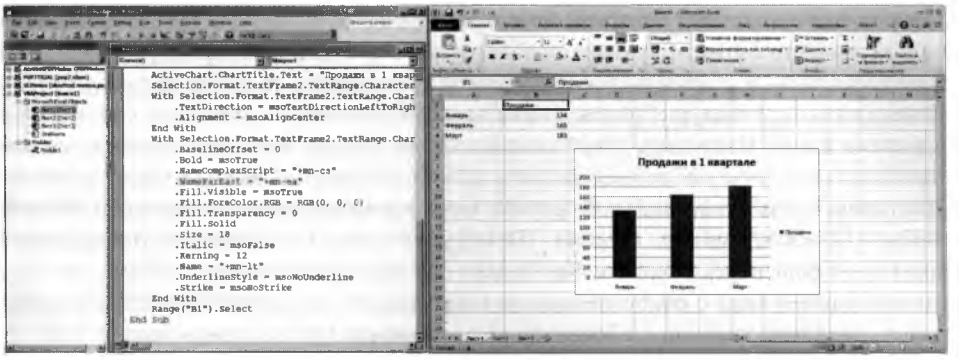


Рис. 7.13. Данное расположение окон облегчает наблюдение за работой функции записи макросов

Абсолютный или относительный

При записи последовательности действий Excel обычно использует *абсолютные ссылки на ячейки*. Другими словами, выделяя ячейку, Excel делает ее активной (а не исходную ячейку, активную при запуске программы). Проверим на примере, как это работает. Выполните следующие действия и проанализируйте полученный результат.

1. Активизируйте рабочий лист и запустите функцию записи макросов.
2. Активизируйте ячейку B1.
3. Введите в ячейку B1 **Янв**.
4. Перейдите в ячейку C1 и введите **Фев**.
5. Продолжайте этот процесс, пока в ячейках B1:G1 не будут введены аббревиатуры первых шести месяцев года.
6. Щелкните на ячейке B1 для ее повторной активизации.
7. Остановите запись макроса и просмотрите полученный код в окне редактора VBE.

Excel сгенерирует следующий код.

```
Sub Macro1()
   Range("B1").Select
   ActiveCell.FormulaR1C1 = "Янв"
   Range("C1").Select
   ActiveCell.FormulaR1C1 = "Фев"
   Range("D1").Select
   ActiveCell.FormulaR1C1 = "Мар"
   Range("E1").Select
   ActiveCell.FormulaR1C1 = "Апр"
   Range("F1").Select
   ActiveCell.FormulaR1C1 = "Май"
   Range("G1").Select
   ActiveCell.FormulaR1C1 = "Июн"
   Range("B1").Select
End Sub
```

Для запуска макроса на выполнение выберите команду Разработчик⇨Код⇨Макросы (Developer⇨Code⇨Macros) либо нажмите комбинацию клавиш <Alt+F8>, выберите `Макрос1` (либо название записанного макроса) и щелкните на кнопке Выполнить (Run).

Макрос вновь выполнит действия, записанные ранее. Действия выполняются независимо от того, активны ли соответствующие ячейки на рабочем листе. При записи макроса с использованием абсолютных ссылок вы всегда будете получать одни и те же результаты.

В некоторых случаях требуется, чтобы записанный макрос работал с *относительными* адресами ячеек. Например, такой макрос обычно вводит названия месяцев в активной ячейке. В таком случае для записи макроса используется относительная форма записи.

Для реализации относительной формы записи воспользуйтесь командой Разработчик⇨Код⇨Относительные ссылки (Developer⇨Code⇨Use Relative References). Эта кнопка играет роль переключателя. Как только она окрашивается в другой цвет, это служит признаком записи кода с относительными ссылками. Если кнопка окрашена в стандартный цвет, записывается код с абсолютными ссылками. Метод записи может быть изменен в любой момент, даже в середине процесса записи.

Чтобы увидеть, как работает относительная форма записи макросов, очистите содержимое ячеек B1:G1 и выполните следующие действия.

1. Активизируйте ячейку B1.
2. Выберите команду Разработчик⇨Код⇨Запись макроса (Developer⇨Code⇨Record Macro).
3. Для начала записи щелкните на кнопке OK.
4. Щелкните на кнопке Относительные ссылки (Use Relative Reference) для выбора относительного режима записи. При этом цвет данной кнопки изменится.
5. В ячейки B1:G1 введите сокращенные названия первых шести месяцев, как в предыдущем примере.
6. Выделите ячейку B1.
7. Остановите запись макроса.

Если установить относительный режим записи, созданный Excel код приобретет иной вид.

```
Sub Macro2()
    ActiveCell.FormulaR1C1 = "Янв"
    ActiveCell.Offset(0, 1).Range("A1").Select
    ActiveCell.FormulaR1C1 = "Фкв"
    ActiveCell.Offset(0, 1).Range("A1").Select
    ActiveCell.FormulaR1C1 = "Мар"
    ActiveCell.Offset(0, 1).Range("A1").Select
    ActiveCell.FormulaR1C1 = "Апр"
    ActiveCell.Offset(0, 1).Range("A1").Select
    ActiveCell.FormulaR1C1 = "Май"
    ActiveCell.Offset(0, 1).Range("A1").Select
    ActiveCell.FormulaR1C1 = "Июн"
    ActiveCell.Offset(0, -5).Range("A1").Select
End Sub
```

Для вызова этого макроса на выполнение активизируйте рабочий лист и выберите команду Разработчик⇨Код⇨Макросы (Developer⇨Code⇨Macros). Выберите имя нужного макроса и щелкните на кнопке Выполнить (Run).

В рассматриваемом примере процедура была незначительно изменена (мы активизировали начальную ячейку *перед* началом записи). Это важная операция при записи макроса, использующего в качестве основы активную ячейку.

Макрос кажется сложным, хотя на самом деле это не так. Первый оператор вводит **Янв** в активную ячейку. (Используется активная ячейка, так как перед оператором не

указан оператор активизации ячейки.) Следующий оператор использует метод `Select` (Выделение) (наравне со свойством `Offset` (Смещение)) для перемещения курсора на одну ячейку вправо. Следующий оператор вставляет в нее текст и т.д. В результате исходная ячейка выделяется путем вычисления относительного адреса (смещения). В отличие от предыдущего, данный макрос всегда начинает ввод текста в активной ячейке.

Функция записи макросов работает в двух различных режимах, причем важно знать, в каком режиме записи макроса вы находитесь в данный момент, иначе результат может не совпадать с ожидаемым.

Примечание

В этом макросе сгенерирован код, который будто бы ссылается на ячейку A1, что может показаться странным, поскольку ячейка A1 в макросе не используется. Это побочный эффект функции записи макросов. (Свойство `Offset` будет рассмотрено далее.) На данном этапе макрос работает так, как требуется.

Кстати, код, сгенерированный Excel, намного сложнее, чем необходимо, и представляет не самый эффективный способ программирования описанной операции. Следующий макрос, который был введен вручную, представляет собой более простой и быстрый способ выполнить те же действия. В примере показано, что не обязательно выделять ячейку перед помещением в нее информации — это важный момент, который существенно ускоряет работу макроса.

```
Sub Macro3()
   ActiveCell.Offset(0, 0) = "Янв"
   ActiveCell.Offset(0, 1) = "Фев"
   ActiveCell.Offset(0, 2) = "Мар"
   ActiveCell.Offset(0, 3) = "Апр"
   ActiveCell.Offset(0, 4) = "Май"
   ActiveCell.Offset(0, 5) = "Июн"
End Sub
```

Данный макрос можно еще больше упростить с помощью конструкции `With-End With`.

```
Sub Macro4()
   With ActiveCell
      .Offset(0, 0) = "Янв"
      .Offset(0, 1) = "Фев"
      .Offset(0, 2) = "Мар"
      .Offset(0, 3) = "Апр"
      .Offset(0, 4) = "Май"
      .Offset(0, 5) = "Июн"
   End With
End Sub
```

Если же вы гениальный программист на VBA (как и я), можете поразить своих коллег, выполнив все описанные выше действия в одном операторе.

```
Sub Macro5()
   ActiveCell.Resize(,6)=Array("Янв","Фев","Мар","Апр","Май","Июн")
End Sub
```

Параметры записи

В процессе записи действий, применяемых для создания VBA-кода, в распоряжении пользователя оказывается ряд параметров диалогового окна Запись макроса (Record Macro). Все они описаны ниже.

- **Имя макроса (Macro Name).** Предоставляет возможность ввести название записываемой процедуры. По умолчанию Excel использует названия `Макрос1`, `Макрос2` и т.д. для каждого записываемого макроса. Вы можете использовать имя по умолчанию и изменить его позже, однако лучше сразу назвать макрос правильным именем.
- **Комбинация клавиш (Shortcut Key).** Позволяет выполнить макрос с помощью комбинации клавиш. Например, введя в данном поле **w** (в нижнем регистре), вы можете выполнить макрос. Для этого нажмите комбинацию клавиш <Ctrl+W>. После ввода символа **W** (в верхнем регистре) макрос запускается по нажатию комбинации клавиш <Ctrl+Shift+W>. Помните о том, что комбинация клавиш, назначенная макросу, переопределяет встроенную комбинацию клавиш (если она есть). Например, если для вызова макроса воспользоваться клавишами <Ctrl+B>, вы не сможете применить эти клавиши для выбора полужирного стиля выделения ячеек.

  Вы вправе в любой момент добавить или изменить комбинацию клавиш, поэтому необязательно задавать параметр при записи макроса.
- **Параметр Сохранить в (Store Macro In).** Указывает Excel, где должен храниться макрос, который записывается. По умолчанию Excel помещает записанный макрос в модуль активной рабочей книги. По желанию можно записать его либо в новой рабочей книге (Excel открывает пустую рабочую книгу), либо в личной книге макросов. (Дополнительные сведения можно найти во врезке "Личная книга макросов".)

Примечание

Excel запоминает ваш выбор, поэтому в следующий раз по умолчанию макрос записывается туда же, куда он записывался в предыдущий раз.

- **Описание (Description).** При желании в поле Описание (Description) можно ввести описание макроса. Введенный текст отображается в начале кода макроса в виде комментария.

Личная книга макросов

В процессе создания макрос можно сохранить в личной книге макросов (Personal Macro Workbook). Здесь можно сохранить макросы VBA, которые вы считаете особенно полезными. Эта рабочая книга называется `Personal.xlsb` и хранится в папке `XLStart`. Загрузка данной рабочей книги происходит после запуска Excel, причем вы сразу же получаете доступ к хранящимся там макросам. Книга `Personal.xlsb` скрыта, поэтому при обычной работе в Excel она не видна.

Файл `Personal.xlsb` не существует до тех пор, пока не будет записан первый макрос.

Улучшение записанных макросов

Итак, вы уже знаете о том, что запись действий при выполнении всего лишь одной команды (Разметка страницы⇨Параметры страницы⇨Ориентация (Page Layout⇨Page Setup⇨Orientation)) приводит к генерированию большого объема кода VBA. Во многих случаях записанный код включает ненужные команды, которые следует удалять вручную.

Кроме того, функция записи макросов не всегда гарантирует получение эффективного кода. Работая с созданной программой, вы можете обнаружить следующее: как правило, Excel анализирует то, что выделено (т.е. определяет активный объект), а затем использует в генерируемых операторах объект `Selection`. Например, ниже приводится пример кода, который генерируется при выделении диапазона ячеек и использовании некоторых кнопок вкладки **Главная** (Home) для изменения числового форматирования, а также применения полужирного и курсивного стилей.

```
Range("A1:C5").Select
Selection.Style = "Comma"
Selection.Font.Bold = True
Selection.Font.Italic = True
```

Записанный код VBA работает, но он представляет собой всего лишь один способ выполнения этих действий. Можно также воспользоваться более эффективной конструкцией `With-End With`, как показано ниже.

```
Range("A1:C5").Select
With Selection
   .Style = "Comma"
   .Font.Bold = True
   .Font.Italic = True
End With
```

Можно избежать применения метода `Select`, в результате чего появится более эффективный код.

```
With Range("A1:C5")
   .Style = "Comma" .Font.Bold = True
   .Font.Italic = True
End With
```

Если в вашем приложении важна скорость выполнения операций, то необходимо тщательно анализировать любой записанный код VBA, чтобы сделать его как можно более эффективным.

Конечно, вам придется разобраться в VBA, прежде чем приступить к улучшению кода записанных макросов. А пока примите к сведению, что записанный код VBA не всегда является наилучшим и наиболее эффективным решением задачи.

О примерах кода в книге

В этой книге представлено много небольших примеров кода VBA, объясняющих отдельную тему или представляющих пример решения поставленной задачи. Зачастую этот код может состоять из единственного оператора. В некоторых случаях примеры представлены только выражением, которое не является корректной инструкцией.

Ниже представлено типичное выражение.

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

Чтобы протестировать выражение, его необходимо выполнить. Для этого часто применяется функция `MsgBox`.

```
MsgBox Range("A1").Value
```

Чтобы выполнить предлагаемые примеры, поместите оператор в процедуру модуля VBA следующим образом.

```
Sub Test()
   ' Здесь находится оператор
End Sub
```

Переместите курсор в любое место процедуры и нажмите клавишу <F5> для ее выполнения. Кроме того, убедитесь, что код выполняется в правильном контексте. Например, если оператор ссылается на лист `Лист1`, удостоверьтесь, что в рабочей книге действительно есть лист с названием `Лист1`.

Код может состоять из единственного оператора. В этом случае используется окно отладки VBE (Immediate). Оно применяется для "немедленного" выполнения операторов — без создания процедуры. Если окно отладки не отображается, нажмите клавиши <Ctrl+G>.

Введите в окно отладки оператор VBA и нажмите клавишу <Enter>. Чтобы проверить выражение в окне отладки, введите перед ним знак вопроса (`?`) — символ вызова команды `Print`. Например, в окно отладки можно ввести следующий код:

```
? Range("A1").Value
```

Результат выполнения выражения отображается в следующей строке окна отладки.

Об объектах и коллекциях

На этом этапе вы должны иметь представление о VBA и знать основные методы управления модулями VBA в редакторе Visual Basic. Кроме того, вы ознакомились с примерами кода VBA и получили достаточно информации о таких элементах, как объекты и свойства. В этом разделе приводится дополнительная информация об объектах и коллекциях объектов.

Работая с кодом VBA, следует четко понимать назначение объектов и объектной модели Excel. Целесообразнее рассматривать объекты с точки зрения *иерархической структуры*. На вершине объектной модели находится объект `Application` (в данном случае — Excel). Но если вы программируете в VBA, запуская VBE в Microsoft Word, то объектом `Application` будет выступать Word.

Иерархия объектов

Объект `Application` (в рассматриваемом случае Excel) содержит другие объекты. Ниже приведено несколько примеров объектов, которые находятся в объекте `Application`:

- `Workbooks` (коллекция всех объектов `Workbook` — рабочих книг);
- `Windows` (коллекция всех объектов `Window` — окон);
- `AddIns` (коллекция всех объектов `AddIn` — надстроек).

Одни объекты могут содержать другие объекты. Например, коллекция `Workbooks` состоит из всех открытых объектов `Workbook`, а объект `Workbook` включает другие объекты. Некоторые из них представлены ниже:

- `Worksheets` (коллекция объектов `Worksheet` — рабочих листов);
- `Charts` (коллекция объектов `Chart` — диаграмм);
- `Names` (коллекция объектов `Name` — имен).

Каждый из этих объектов, в свою очередь, может содержать другие объекты. Коллекция `Worksheets` состоит из всех объектов `Worksheet` рабочей книги `Workbook`. Объект `Worksheet` включает другие объекты, среди которых есть следующие:

- `ChartObjects` (коллекция объектов `ChartObject` — элементов диаграмм);
- `Range` — диапазон;
- `PageSetup` — параметры страницы;
- `PivotTables` (коллекция объектов `PivotTable` — сводных таблиц).

Может быть, вы пока не готовы правильно воспринять подобную концепцию, но со временем наверняка поймете, что иерархия объектов вполне логична и хорошо структурирована. Вся объектная модель Excel схематически изображена в электронной справочной системе.

О коллекциях

Одной из ключевых концепций в программировании на языке VBA является коллекция. *Коллекция* — это группа объектов одного класса (и сама коллекция также является объектом). Как указывалось выше, `Workbooks` — это коллекция всех открытых в данный момент объектов `Workbook`. `Worksheets` — коллекция всех объектов `Worksheet`, которые содержатся в конкретном объекте `Workbook`. Можно одновременно управлять целой коллекцией объектов или отдельным объектом этой коллекции. Чтобы сослаться на один объект из коллекции, введите название или номер объекта в скобках после названия коллекции.

```
Worksheets("Лист1")
```

Если лист `Лист1` — это первый рабочий лист в коллекции, то можно использовать следующую ссылку:

```
Worksheets(1)
```

Если в объекте `Workbook` нужно сослаться на второй рабочий лист, ссылка указывается в формате `Worksheets(2)` и т.д.

Коллекция `Sheets` состоит из всех листов рабочей книги, которые могут представлять собой рабочие листы или листы диаграмм. Если `Лист1` — первый лист в рабочей книге, ссылка на него имеет следующий вид:

```
Sheets(1)
```

Ссылки на объекты

Если вы обращаетесь к объекту в VBA, то в ссылке на него вводятся названия всех расположенных выше в иерархической структуре объектов, разделенных точкой (*оператор-точка*). Что делать, если в Excel открыты две рабочие книги и в обеих имеется рабочий лист с названием `Лист1`? В этом случае в ссылке необходимо указать контейнер требуемого объекта.

```
Workbooks("Книга1").Worksheets("Лист1")
```

Без указания рабочей книги редактор Visual Basic искал бы `Лист1` в активной рабочей книге.

Чтобы сослаться на определенный диапазон (например, на ячейку A1) на рабочем листе с названием `Лист1` в рабочей книге `Книга1`, можно использовать следующее выражение:

```
Workbooks("Книга1").Worksheets("Лист1").Range("A1")
```

Полная ссылка из предыдущего примера включает объект `Application` и выглядит так:

```
Application.Workbooks("Книга1").Worksheets("Лист1").Range("A1")
```

Однако в большинстве случаев можно опускать объект Application в ссылках (кроме него использоваться больше нечему). Если объект Книга1 — это активная рабочая книга, то опустите ссылку на нее и запишите рассматриваемое выражение следующим образом:

```
Worksheets("Книга1").Range("A1")
```

Если Лист1 — активный рабочий лист, можно еще более упростить выражение:

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

Примечание

В Excel отсутствует объект отдельной ячейки. Отдельная ячейка представляет собой объект Range, состоящий из одного элемента.

Простые ссылки на объекты (как в приведенных примерах) ничего не выполняют. Чтобы выполнить действие, прочтите или измените свойства объекта или задайте метод, который выполняется по отношению к объекту.

Свойства и методы

Запутаться в свойствах и методах несложно: их существует несколько тысяч. В этом разделе показано, как осуществляется доступ к свойствам и методам объектов.

Свойства объекта

Все объекты обладают свойствами. Например, объект Range (Диапазон) обладает свойством Value (Значение). Можно создать оператор VBA, чтобы отобразить свойство Value или присвоить ему определенное значение. Ниже приведена процедура, использующая функцию VBA MsgBox для отображения окна, в котором представлено значение ячейки A1 листа Лист1 активной рабочей книги.

```
Sub ShowValue()
   Msgbox Worksheets("Лист1").Range("A1").Value
End Sub
```

Примечание

Функция VBA MsgBox обеспечивает простой способ отображения результатов выполнения кода VBA. Эта функция повсеместно используется в книге.

Код предыдущего примера отображает текущее значение свойства Value для конкретной ячейки — A1 рабочего листа Лист1 активной рабочей книги. Обратите внимание на то, что если в активной книге отсутствует лист с названием Лист1, то макрос выдаст ошибку.

Что необходимо сделать, чтобы изменить свойство Value? Ниже приведена процедура изменения значения ячейки A1 путем определения значения свойства Value.

```
Sub ChangeValue()
   Worksheets("Лист1").Range("A1").Value = 123.45
End Sub
```

После выполнения этой процедуры ячейка A1 листа Лист1 получает значение 123,45.

Можете ввести описанные процедуры в модуль и протестировать их.

Примечание

Многие объекты имеют свойство, заданное по умолчанию. Для объекта Range свойством по умолчанию является Value. Следовательно, выражение .Value в предыдущем коде можно опустить, и ничего не произойдет. Однако лучше включать ссылку на свойство, даже если оно используется по умолчанию.

Ниже приводится выражение, в котором доступ к значениям осуществляется с помощью свойств HasFormula и Formula объекта Range.

```
If Range("A1").HasFormula Then MsgBox Range("A1").Formula
```

В данном случае конструкция If-Then применяется для условного отображения окна сообщения: если ячейка содержит формулу, эта формула будет отображена путем обращения к свойству Formula. Если ячейка A1 не содержит формулу, ничего не происходит.

Свойство Formula доступно в режиме "только для чтения", поэтому определить формулу можно с помощью кода VBA.

```
Range("D12").Formula = "=RAND()*100"
```

Методы объекта

Кроме свойств, объекты характеризуются методами. *Метод* — это действие, которое выполняется над объектом. Ниже приведен простой пример использования метода Clear по отношению к диапазону ячеек. После выполнения этой процедуры ячейки A1:C3 листа Лист1 станут пустыми, и дополнительное их форматирование будет удалено.

```
Sub ZapRange()
   Worksheets("Лист1").Range("A1:C3").Clear
End Sub
```

Если нужно удалить значения в диапазоне, но оставить форматирование, используйте метод ClearContents объекта Range.

Многие методы получают аргументы, определяющие выполняемые над объектом действия более детально. Далее приводится пример, в котором ячейка A1 копируется в ячейку B1 с помощью метода Copy объекта Range. В данном примере метод Copy получает один аргумент (адрес ячейки, в которую следует скопировать данные). Обратите внимание, что в примере используется символ продолжения строки (пробел и подчеркивание). Можно не применять этот символ, а ввести оператор в одну строку.

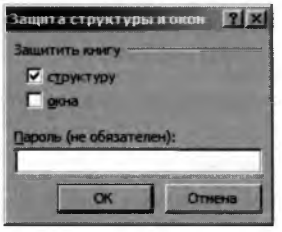
```
Sub CopyOne()
   Worksheets("Лист1").Range("A1").Copy _
      Worksheets("Лист1").Range("B1")
End Sub
```

Определение аргументов методов и свойств

В среде программистов VBA определение аргументов методов и свойств часто вызывает трудности. Некоторые методы используют аргументы для дальнейшего уточнения действий, отдельные свойства применяют аргументы для последующего определения значения свойства. Иногда один или несколько аргументов вообще применять необязательно.

Если метод использует аргументы, они указываются после названия метода и разделяются запятыми. Если метод использует необязательные аргументы, можете пропустить их, оставив пустые места.

А теперь обратимся к методу `Protect` объекта рабочей книги. В справочной системе можно найти информацию о том, что метод `Protect` имеет три аргумента: пароль, структуру и окна. Эти аргументы соответствуют параметрам диалогового окна Защита книги (Protect Workbook).



Например, если нужно защитить рабочую книгу `MyBook.xlsx`, используйте следующий оператор:

```
Workbooks("MyBook.xlsx").Protect "xyzzy", True, False
```

В данном случае рабочая книга защищена паролем (аргумент 1). Также защищена структура (аргумент 2), но не окна (аргумент 3).

Если вы не хотите применять пароль, используйте следующий оператор:

```
Workbooks("MyBook.xlsx").Protect , True, False
```

Обратите внимание, что первый аргумент пропущен, а его место обозначено запятой.

Существует и другой подход (причем в этом случае программу будет удобнее читать) — использование именованных аргументов. Применим именованные аргументы для предыдущего примера.

```
Workbooks("MyBook.xlsx").Protect Structure:=True, Windows:=False
```

Применение именованных аргументов — хорошая идея, особенно в методах с большим количеством необязательных аргументов, когда нужно использовать только некоторые из них. При работе с именованными аргументами не нужно оставлять место для отсутствующих аргументов.

Для свойств (и методов), использующих аргументы, последние указываются в скобках. Например, свойство `Address` объекта `Range` имеет пять аргументов — все необязательные. Показанный ниже оператор некорректен, поскольку пропущены скобки.

```
MsgBox Range("A1").Address False ' Некорректно
```

Правильный синтаксис для этого оператора выглядит так:

```
MsgBox Range("A1").Address(False)
```

Кроме того, оператор можно записать с помощью именованного аргумента.

```
MsgBox Range("A1").Address(rowAbsolute:=False)
```

Подобные нюансы станут для вас понятнее по мере приобретения опыта программирования в VBA.

Объект `Comment`: пример использования

Чтобы лучше разобраться в свойствах и методах объекта, сосредоточимся на изучении конкретного объекта — `Comment`. Этот объект можно создать с помощью команды Excel Рецензирование⇨Примечания⇨Создать примечание (Review⇨Comments⇨New Comment). Его назначение — вставка комментария в ячейки. В следующих разделах вы получите более детальное представление об управлении этим объектом.

Справочные сведения об объекте `Comment`

Единственный способ получить информацию о конкретном объекте — найти соответствующие разделы в электронной справочной системе. На рис. 7.14 показано главное окно справочной системы для объекта `Comment`. Для отображения этого экрана было

введено слово **comment** в поле Type a Question (Задайте вопрос) окна справки VBE (в правой части строки меню). Обратите внимание, что в содержании справки отображаются свойства и методы этого объекта.

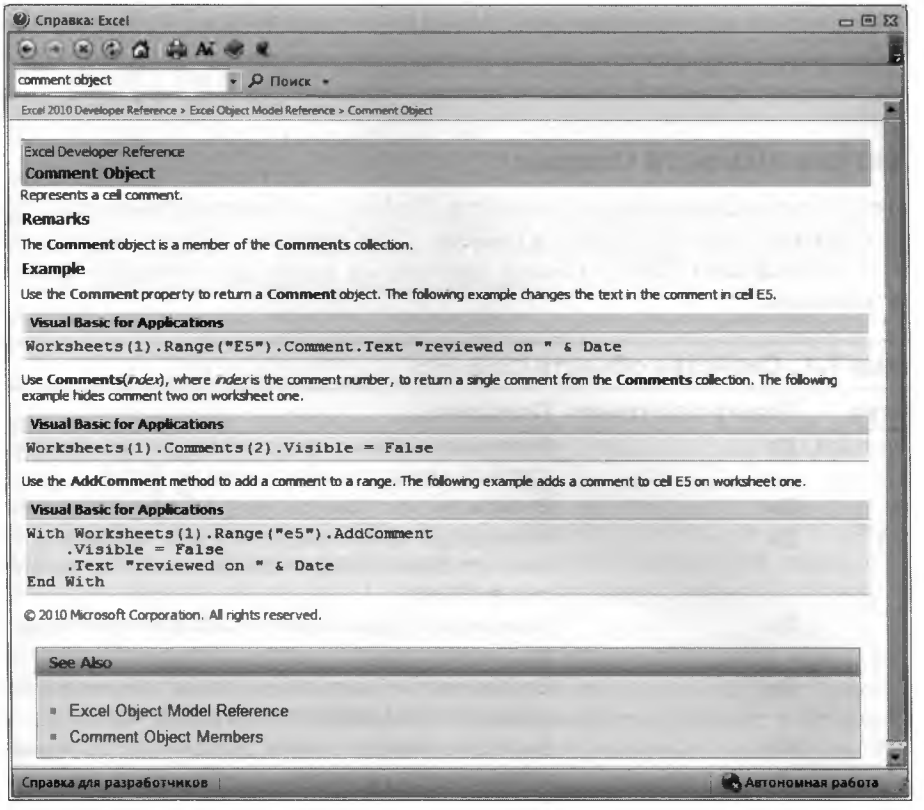


Рис. 7.14. Главное окно справки для объекта Comment

Использование электронной справочной системы

Самый простой способ получить справку о конкретном объекте, свойстве или методе — ввести ключевое слово в окно кода и нажать клавишу <F1>. Если ключевое слово трактуется неоднозначно, появляется диалоговое окно выбора темы, подобное показанному на следующем рисунке.

К сожалению, элементы, перечисленные в этом диалоговом окне, не всегда понятны, поэтому, чтобы найти нужный раздел, часто приходится прибегать к методу проб и ошибок. Показанное на рисунке диалоговое окно появилось в результате ввода слова `Comment` с последующим нажатием клавиши <F1>. В рассматриваемом случае `Comment` является объектом, который может вести себя подобно свойству. После щелчка на первой теме отображается раздел, посвященный объекту `Comment`; если выбрать вторую тему, отображается раздел, посвященный свойству `Comment`.

Свойства объекта `Comment`

Объект `Comment` включает шесть свойств. Список этих свойств приводится в табл. 7.1 наравне с кратким описанием каждого свойства. Если свойство доступно только в режиме чтения, созданный вами VBA-код может применяться только для чтения свойства, но не для его изменения.

Таблица 7.1. Свойства объекта `Comment`

Свойство	Только для чтения	Описание
`Application`	Да	Возвращает объект, представляющий приложение, в котором создавалось примечание (т.е. Excel)
`Author`	Да	Возвращает имя пользователя, создавшего примечание
`Creator`	Да	Возвращает целое число, которое свидетельствует о том, что было создано приложение, к которому относится объект
`Parent`	Да	Возвращает родительский объект для примечания (всегда объект `Range`)
`Shape`	Да	Возвращает объект `Shape`, представляющий фигуру, в виде которой отображается примечание
`Visible`	Нет	Если это свойство имеет значение `True` примечание отображается на экране

Методы объекта `Comment`

В табл. 7.2 приведены методы, которые можно использовать в объекте `Comment`. Все они выполняют обычные операции, которые производятся над примечанием вручную, однако прежде мы не рассматривали эти действия как методы.

Таблица 7.2. Методы объекта `Comment`

Метод	Описание
`Delete`	Удаляет комментарий
`Next`	Возвращает объект `Comment`, который представляет следующий комментарий в рабочем листе
`Previous`	Возвращает объект `Comment`, который представляет предыдущий комментарий в рабочем листе
`Text`	Возвращает или определяет текст в комментарии (метод имеет три аргумента)

Примечание

Возможно, вас удивило, что `Text` — это метод, а не свойство. Данный формат приводит нас к важному умозаключению: различия между свойствами и методами не всегда четкие, а объектная модель не является идеально последовательной. На самом деле неважно, насколько точно вы будете различать свойства и методы. Пока используется правильный синтаксис, не имеет значения, какую роль в коде выполняет ключевое слово — свойства или метода.

Коллекция `Comments`

Коллекция — это группа одинаковых объектов. Каждый рабочий лист имеет коллекцию `Comments`, состоящую из всех объектов `Comment` рабочего листа. Если на рабочем листе отсутствуют примечания, эта коллекция пуста.

Например, приведенный ниже код ссылается на первое примечание листа `Лист1` активной рабочей книги.

```
Worksheets("Лист1").Comments(1)
```

Следующий оператор отображает текст, который содержится в первом примечании листа `Лист1`:

```
MsgBox Worksheets("Лист1").Comments(1).Text
```

В отличие от большинства объектов, объект `Comment` не имеет свойства `Name`. Следовательно, чтобы сослаться на конкретный комментарий, используйте номер, а для получения необходимого комментария обратитесь к свойству `Comment` объекта `Range` либо укажите значение индекса.

Коллекция `Comments` — тоже объект, имеющий собственный набор свойств и методов. Например, коллекция `Comments` включает свойство `Count`, которое хранит сведения о количестве элементов в коллекции, — число элементов `Comment` в активной рабочей книге. Так, следующий пример определяет общее количество комментариев:

```
MsgBox ActiveSheet.Comments.Count
```

В следующем примере показан адрес ячейки, в которой находится первое примечание:

```
MsgBox ActiveSheet.Comments(1).Parent.Address
```

В этом примере `Comments(1)` возвращает первый объект `Comment` коллекции `Comments`. Свойство `Parent` объекта `Comment` возвращает его контейнер, представленный объектом `Range`. В окне сообщений отображается свойство `Address` объекта `Range`. В итоге оператор показывает адрес ячейки, содержащей первое примечание.

Кроме того, вы можете циклически просмотреть все примечания на листе, используя конструкцию `For Each-Next` (о ней речь пойдет в главе 8). Ниже приведен пример использования отдельных сообщений для раздельного отображения каждого примечания активного рабочего листа.

```
For Each cmt in ActiveSheet.Comments
   MsgBox cmt.Text
Next cmt
```

Если вы не хотите, чтобы на экране находилось большое количество диалоговых окон с сообщениями, то используйте следующую процедуру для вывода всех примечаний в одном окне отладки (Immediate).

```
For Each cmt in ActiveSheet.Comments
   Debug.Print cmt.Text
Next cmt
```

О свойстве Comment

В этом разделе речь идет об объекте Comment. В справочной системе указано, что объект Range обладает свойством Comment. Если ячейка содержит примечание, свойство Comment возвращает объект Comment. Например, следующий оператор ссылается на объект Comment ячейки A1:

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

Если это первое примечание на листе, то на данный объект Comment можно сослаться следующим образом:

```
ActiveSheet.Comments(1)
```

Чтобы отобразить примечание ячейки A1 в окне сообщения, используйте такой оператор:

```
MsgBox Range("A1").Comment.Text
```

Если в ячейке A1 нет примечания, то оператор выдаст ошибку.

Примечание

Тот факт, что свойство может возвращать объект, довольно важен — данная концепция имеет решающее значение в программировании на VBA.

Объекты, вложенные в Comment

Управление свойствами сначала кажется сложной задачей, потому что некоторые из них возвращают объекты. Предположим, необходимо определить цвет фона конкретного примечания на листе Лист1. Просмотрев список свойств объекта Comment, вы не найдете ничего, что относится к определению цвета. Выполните следующие действия.

1. Используйте свойство Shape объекта Comment, возвращающее объект Shape, который содержится в примечании.
2. Используйте свойство Fill объекта Shape, возвращающее объект FillFormat.
3. Используйте свойство ForeColor объекта FillFormat, возвращающее объект ColorFormat.
4. Используйте свойство RGB (или свойство SchemeColor) объекта ColorFormat, чтобы задать цвет.

Иначе говоря, получение цвета фона объекта Comment связано с доступом к другим объектам, которые в нем содержатся. Ниже описана иерархия задействованных объектов.

```
Application (Excel)
   Объект Workbook
      Объект Worksheet
         Объект Comment
            Объект Shape
               Объект FillFormat
                  Объект ColorFormat
```

Данная иерархия объектов выглядит весьма запутанной! Но в качестве примера "элегантности" VBA посмотрите, как код для изменения цвета примечания можно записать с помощью одного оператора.

```
Worksheets("Лист1").Comments(1).Shape.Fill.ForeColor _
   .RGB = RGB(0, 255, 0)
```

Вы вправе использовать также свойство SchemeColor (значение которого определяется в диапазоне от 0 до 80).

```
Worksheets("Лист1").Comments(1).Shape.Fill.ForeColor _
   .SchemeColor = 12
```

В данном типе ссылки сразу трудно разобраться, но впоследствии вам несложно будет ориентироваться в иерархии объектов. Вы быстро изучите ее особенности, прежде всего потому, что в Excel при записи последовательности действий практически всегда вопрос иерархии задействованных объектов ставится на первое место.

Кстати, чтобы изменить цвет текста в примечании, обратитесь к объекту TextFrame объекта Comment, который содержит объект Characters, включающий, в свою очередь, объект Font. Далее обратитесь к свойству Color или ColorIndex объекта Font.

Ниже приведен пример, в результате выполнения которого свойству ColorIndex присваивается значение 5.

```
Worksheets("Лист1").Comments(1) _
   .Shape.TextFrame.Characters.Font.ColorIndex = 5
```

Перекрестная ссылка

Обратитесь к главе 30 для получения дополнительных сведений о цветах.

Содержит ли ячейка примечание

Следующий оператор отображает примечание ячейки A1 активного листа:

```
MsgBox Range("A1").Comment.Text
```

Если в ячейке A1 примечание отсутствует, при выполнении этого оператора возникнет непонятное сообщение об ошибке: Object variable or With block variable not set (Не присвоено значение объектной переменной или переменной блока With).

Чтобы определить, содержит ли конкретная ячейка примечание, напишите код, проверяющий, не пустой ли объект Comment, т.е. равен ли он Nothing (это корректное ключевое слово VBA). Следующий оператор отображает True, если в ячейке A1 примечание отсутствует:

```
MsgBox Range("A1").Comment Is Nothing
```

Обратите внимание, что в этом примере используется ключевое слово Is, а не знак равенства.

Вы можете сделать на один шаг больше и создать оператор, который отображает примечание ячейки только в том случае, если это примечание находится в данной ячейке (при этом не появляется сообщение об ошибке в случае отсутствия примечания). Соответствующий оператор приводится ниже.

```
If Not Range("A1").Comment Is Nothing Then _
   MsgBox Range("A1").Comment.Text
```

В данном случае использовалось ключевое слово Not, которое инвертирует значение True, возвращаемое, если ячейка не содержит примечание. Фактически этот оператор использует двойное отрицание в проверочном условии. Если примечание не отсутствует, оно отображается.

Добавление нового объекта Comment

Возможно, вы заметили, что в списке методов объекта Comment не существует метода для добавления нового примечания. Это объясняется тем, что метод AddComment принадлежит объекту Range. Следующий оператор добавляет примечание (пустое) в ячейку A1 активного рабочего листа:

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

Обратившись к справочной системе, вы обнаружите, что метод AddComment имеет аргумент, представляющий текст примечания. Следовательно, можно добавить примечание и текст в нем с помощью всего одного оператора.

```
Range("A1").AddComment "Формула разработана JW."
```

Примечание

Метод AddComment генерирует ошибку, если в ячейке уже содержится примечание. Во избежание появления этой ошибки проверьте содержимое ячейки.

Компакт-диск

Рассмотренные свойства и методы объекта Comment можно увидеть в действии (см. пример на прилагаемом к книге компакт-диске). Соответствующая рабочая книга (под именем Commentobject.xlsm) содержит несколько примеров управления объектами Comment с помощью кода VBA. Этот код поможет вам осознать, как можно использовать VBA для управления объектами.

Некоторые полезные свойства объекта Application

При работе в Excel активной одновременно может быть только одна рабочая книга. И если вы управляете рабочим листом, то активна на нем только одна ячейка (даже если выделен диапазон). VBA это известно, поэтому вы можете ссылаться на активные объекты более простым методом. Это удобно, так как не всегда известно, с какой именно рабочей книгой, рабочим листом или ячейкой вам предстоит работать. VBA представляет свойства объекта Application для определения этого. Например, объект Application обладает свойством ActiveCell, возвращающим ссылку на активную ячейку. Следующая инструкция присваивает значение 1 активной ячейке:

```
ActiveCell.Value = 1
```

Обратите внимание, что в этом примере пропущена ссылка на объект Application, так как это само собой разумеется. Важно понять, что такая инструкция может выдать ошибку, если активный лист не является рабочим. Например, если VBA выполняет этот оператор, когда активен лист диаграммы, то процедура прекращает выполняться, а на экране отображается сообщение об ошибке.

Если на рабочем листе выделен диапазон ячеек, то активная ячейка будет находиться в выделенном диапазоне. Другими словами, активная ячейка всегда одна (их никогда не бывает несколько).

Объект Application также обладает свойством Selection, возвращающим ссылку на выделенный объект, т.е. отдельную ячейку (активную), диапазон ячеек или объект типа ChartObject, TextBox, Shape.

В табл. 7.3 перечислены свойства объекта Application, которые вам пригодятся при работе с ячейками и диапазонами ячеек.

Таблица 7.3. Некоторые полезные свойства объекта Application

Свойство	Возвращаемый объект
ActiveCell	Активная ячейка
ActiveChart	Активный лист диаграмм или объект диаграммы на рабочем листе (ChartObject). Если диаграмма не активна, то свойство равно Nothing
ActiveSheet	Активный лист (рабочий лист или лист диаграммы)
ActiveWindow	Активное окно
ActiveWorkbook	Активная рабочая книга
Selection	Выделенный объект (объект Range, Shape, ChartObject и т.д.)
ThisWorkbook	Рабочая книга, содержащая выполняемую процедуру VBA

Преимущество использования этих свойств для получения объекта заключается в том, что совершенно не обязательно знать, какая ячейка, рабочий лист или книга является активной, и вводить конкретную ссылку на этот объект. Данный факт позволяет создавать код VBA, который не ограничивается конкретной книгой, листом или ячейкой. Например, следующая инструкция удаляет содержимое активной ячейки, даже если адрес активной ячейки не известен:

```
ActiveCell.ClearContents
```

В следующем примере отображается сообщение, указывающее имя активного листа:

```
MsgBox ActiveSheet.Name
```

Если требуется узнать название активной рабочей книги, используйте такой оператор:

```
MsgBox ActiveWorkbook.FullName
```

Если на рабочем листе выделен диапазон, то заполните последний одним значением, выполнив единственный оператор. В следующем примере свойство Selection объекта Application возвращает объект Range, соответствующий выделенным ячейкам. Оператор изменяет свойство Value этого объекта Range, и в результате получается диапазон, заполненный одним значением.

```
Selection.Value = 12
```

Обратите внимание: если выделен не диапазон ячеек (например, объект ChartObject или Shape), то этот оператор выдаст ошибку, так как объекты ChartObject и Shape не обладают свойством Value.

Однако приведенный ниже оператор присваивает объекту Range, который выделялся перед выделением другого объекта (отличного от диапазона ячеек), значение 12. В справочной системе указано, что свойство RangeSelection относится только к объекту Window.

```
ActiveWindow.RangeSelection.Value = 12
```

Чтобы узнать, сколько ячеек выделено на рабочем листе, применяется свойство Count. Ниже приводится соответствующий пример.

```
MsgBox ActiveWindow.RangeSelection.Count
```

Работа с объектами Range

Работа, которая выполняется в VBA, в основном связана с управлением ячейками и диапазонами на рабочих листах, что и является главным предназначением электронных таблиц. Вопрос управления ячейками в VBA рассматривался при обсуждении относительного и абсолютного способов записи макросов, однако ему необходимо уделить особое внимание.

Объект Range содержится в объекте Worksheet и состоит из одной ячейки или диапазона ячеек на отдельном рабочем листе. В следующих разделах будут рассмотрены три способа задания ссылки на объекты Range в программе VBA:

- свойство Range объекта класса Worksheet или Range;
- свойство Cells объекта Worksheet;
- свойство Offset объекта Range.

Свойство Range

Свойство Range возвращает объект Range. Из справочных сведений о свойстве Range вы узнаете, что к нему можно обратиться с помощью нескольких вариантов синтаксиса.

```
объект.Range(ячейка1)
объект.Range(ячейка1, ячейка2)
```

Работа с объединенными ячейками

Работа с объединенными ячейками может вызывать затруднения. Если диапазон содержит такие ячейки, придется выполнять некоторые специальные действия с помощью макроса. Например, если ячейки A1:D1 объединены, показанный ниже оператор выделит все столбцы от A до D (а не только столбец B, как можно ошибочно полагать).

```
Columns("B:B").Select
```

Возможно, подобное поведение является намеренным либо результатом какой-либо ошибки. В любом случае оно может вызвать непредвиденные реакции макроса. Объединенные ячейки также приводят к появлению проблем в процессе сортировки.

Для определения объединенных ячеек, содержащихся в диапазоне, можно использовать приведенную ниже функцию VBA. Эта функция возвращает значение True, если в диапазоне содержится хотя бы одна объединенная ячейка (дополнительные сведения о процедурах-функциях можно найти в главе 10).

```
Function ContainsMergedCells(rng As Range)
   im cell As Range
   ontainsMergedCells = False
   For Each cell In rng
      If cell.MergeCells Then
         ContainsMergedCells = True
         Exit Function
      End If
```

```
    Next cell
End Function
```

Для создания ссылки на объединенные ячейки можно сослаться на весь диапазон объединенных ячеек либо на верхнюю левую ячейку в объединенном диапазоне. Например, если рабочий лист включает четыре ячейки, объединенные в одну (A1, B1, A2 и B1), сослаться на объединенные ячейки можно с помощью одного из следующих выражений.

```
Range("A1:B2")
Range("A1")
```

Если попытаться присвоить значение ячейке в объединенном диапазоне, причем эта ячейка не является верхней левой, VBA игнорирует эту инструкцию и даже не генерирует сообщение об ошибке. Например, следующий оператор бесполезен, если ячейки A1:B2 объединены:

```
Range("B2").Value = 43
```

Некоторые операции приводят к тому, что Excel отображает подтверждающее сообщение. Например, если ячейки A1:B2 объединены, следующий оператор генерирует сообщение `This operation will cause some merged cells to unmerge. Do you wish to continue?` (Эта операция может привести к отмене статуса некоторых объединенных ячеек. Продолжать?).

```
Range("B2").Delete
```

Мораль? Будьте осторожны при работе с объединенными ячейками. Перед применением этого свойства хорошо подумайте.

Свойство `Range` относится к одному из двух типов объектов: `Worksheet` или `Range`. В данном случае `ячейка1` и `ячейка2` указывают параметры, которые Excel будет воспринимать как идентифицирующие диапазон (в первом случае) или очерчивающие диапазон (во втором случае). Ниже следует несколько примеров использования метода `Range`.

В данной главе уже рассматривались примеры, подобные представленным ниже. Далее приведена инструкция, которая вводит значение в указанную ячейку: значение **`12,3`** вводится в ячейку A1 на листе `Лист1` активной рабочей книги.

```
Worksheets("Лист1").Range("A1").Value = 12.3
```

Свойство `Range` также поддерживает имена, определенные в рабочих книгах. Поэтому, если ячейка называется `Ввод`, то для ввода значения в нее может использоваться следующий оператор:

```
Worksheets("Лист1").Range("Ввод").Value = 100
```

В следующем примере в диапазон из двадцати ячеек на активном листе вводится одинаковое значение. Если активный лист не является рабочим, то отображается сообщение об ошибке.

```
ActiveSheet.Range("A1:B10").Value = 2
```

Приведенный ниже пример приведет к тому же результату, что и предыдущий.

```
Range("A1", "B10") = 2
```

Отличие заключается лишь в том, что опущена ссылка на лист, поэтому предполагается активный рабочий лист. Кроме того, пропущено свойство, поэтому используется свойство по умолчанию (для объекта `Range` это свойство `Value`). В приведенном примере используется второй синтаксис ссылки на свойство `Range`. В данном случае пер-

вый аргумент — это левая верхняя ячейка диапазона, а второй — ячейка в правом нижнем углу диапазона.

В следующем примере для получения пересечения двух диапазонов применяется оператор пересечения Excel (пробел). Пересечением является одна ячейка — С6. Следовательно, данный оператор вводит значение **3** в ячейку С6.

```
Range("C1:C10 A6:E6") = 3
```

Наконец, в следующем примере значение **4** вводится в пять ячеек, т.е. в независимые диапазоны. Запятая выполняет роль оператора объединения.

```
Range("A1,A3,A5,A7,A9") = 4
```

До настоящего момента во всех рассмотренных примерах использовалось свойство `Range` объекта `Worksheet`. Вы также можете использовать свойство `Range` объекта `Range`. Сначала будет непросто, однако постарайтесь разобраться.

Ниже показан пример использования свойства `Range` объекта `Range` (в данном случае объектом `Range` является активная ячейка). В этом примере объект `Range` рассматривается как левая верхняя ячейка на рабочем листе. В ячейку, которая в таком случае была бы В2, вводится значение **5**. Другими словами, полученная ссылка является относительной для верхнего левого угла объекта `Range`. Таким образом, следующий оператор вводит значение **5** в ячейку, расположенную справа внизу от активной ячейки.

```
ActiveCell.Range("B2") = 5
```

Существует также намного более понятный способ обратиться к ячейке по отношению к диапазону — использовать свойство `Offset` (рассмотрено далее).

Свойство `Cells`

Другим способом сослаться на диапазон является применение свойства `Cells`. Подобно `Range`, свойство `Cells` может использоваться в объектах `Worksheet` и `Range`. Справочная система указывает на три варианта синтаксиса свойства `Cells`.

```
объект.Cells(номер_строки, номер_столбца)
объект.Cells(номер_строки)
объект.Cells
```

Проиллюстрируем на примерах особенности применения свойства `Cells`. Вначале в ячейку А1 листа `Лист1` введите значение **9**. В данном случае используется первый синтаксис, в котором аргументами являются номер строки (от 1 до 1048576) и номер столбца (от 1 до 16384).

```
Worksheets("Лист").Cells(1, 1) = 9
```

Ниже приведен пример, в котором значение **7** вводится в ячейку D3 (т.е. пересечение строки 3 и столбца 4) активного рабочего листа.

```
ActiveSheet.Cells(3, 4) = 7
```

Можно также использовать свойство `Cells` объекта `Range`. При этом объект `Range`, который возвращается свойством `Cells`, задается относительно левой верхней ячейки диапазона `Range`, на который мы ссылаемся. Сложно? Может быть. Приведем пример. Следующая инструкция вводит значение **5** в активную ячейку. Помните, что в данном случае активная ячейка рассматривается как ячейка А1 на рабочем листе.

```
ActiveCell.Cells(1, 1) = 5
```

Примечание

Подлинное преимущество описанного выше типа ссылок на ячейки станет очевидным, когда речь пойдет о переменных и циклах (глава 8). В большинстве случаев в качестве значений аргументов используются не фактические величины, а переменные.

Чтобы ввести значение 5 в ячейку, которая находится под активной, обратитесь к такой инструкции:

```
ActiveCell.Cells(2, 1) = 5
```

Предыдущий пример можно описать так: необходимо начать с активной ячейки, рассматривая ее как ячейку A1. Затем обратитесь к ячейке во второй строке и первом столбце диапазона.

Еще один синтаксис свойства `Cells` использует один аргумент, который задается в диапазоне от 1 до 17179869184. Второе число равно количеству ячеек на рабочем листе Excel 2010. Ячейки нумеруются, начиная с A1 вправо, затем вниз и вправо вдоль следующей строки. Ячейка 16384 — это XFD1, а 16385 — A2.

Далее в ячейку SZ1 активного листа (ячейку 520 на рабочем листе) введем значение 2.

```
ActiveSheet.Cells(520) = 2
```

Для отображения значения последней ячейки рабочего листа (XFD1048576) воспользуйтесь следующим оператором:

```
MsgBox ActiveSheet.Cells(17179869184)
```

Этот синтаксис можно использовать и с объектом `Range`. В таком случае будет получена ячейка по отношению к указанному объекту `Range`. Например, если объект `Range` — это диапазон A1:D10 (40 ячеек), то свойство `Cells` может иметь аргумент от 1 до 40 и возвращать одну из ячеек объекта `Range`. В следующем примере значение **2000** вводится в ячейку A2, так как A2 является пятой ячейкой (считая сверху направо и вниз) в указанном диапазоне.

```
Range("A1:D10").Cells(5) = 2000
```

Получение информации из ячейки

В случае необходимости получения содержимого ячейки обратитесь к соответствующим свойствам VBA. Ниже приведены те из них, которые используются наиболее часто.

- Свойство `Formula` возвращает формулу в случае, когда она находится в ячейке. Если ячейка не содержит формулу, возвращается находящееся в ней значение. Свойство `Formula` может как считываться, так и изменяться пользователем, а также выражаться несколькими вариантами, среди которых `FormulaR1C1`, `FormulaLocal` и `FormulaArray`. (Дополнительные сведения по этом теме можно найти в справочной системе.)
- Свойство `Value` возвращает исходное неотформатированное значение ячейки. Это свойство может считываться и изменяться пользователем.
- Свойство `Text` возвращает текст, отображаемый в ячейке. Если ячейка содержит числовое значение, это свойство включает все имеющееся форматирование, включая запятые и денежные символы. Свойство `Text` предназначено только для чтения.
- Свойство `Value2` подобно свойству `Value`, но не использует типы данных `Date` и `Currency`. Свойство `Value2` преобразует типы данных `Date` и `Currency` в тип

данных Variant, содержащий значения удвоенной точности. Например, если в ячейке находится значение даты 12/6/2010, свойство Value вернет ее как Date, а свойство Value2 — в виде значения удвоенной точности (например, 40518).

Примечание

В предыдущем примере аргумент свойства Cells не ограничен значениями в диапазоне от 1 до 40. Если значение аргумента превышает количество ячеек в диапазоне, счет продолжается, как будто он больше, чем есть на самом деле. Следовательно, оператор, подобный предыдущему, может изменить значение ячейки, которая находится за пределами указанного диапазона A1:D10. Например, следующий оператор изменяет значение в ячейке A11:

```
Range("A1:D10").Cells(41)=2000
```

Третий пример синтаксиса свойства Cells возвращает все ячейки на указанном рабочем листе. В отличие от двух других, в этом синтаксисе возвращаемыми в результате данными будет не одна ячейка, а целый диапазон. В приведенном ниже примере использован метод ClearContents по отношению к диапазону, полученному с помощью свойства Cells для активного рабочего листа. В результате будет удалено содержимое каждой ячейки на рабочем листе.

```
ActiveSheet.Cells.ClearContents
```

Свойство Offset

Свойство Offset (подобно свойствам Range и Cells) также возвращает объект Range. В отличие от рассмотренных выше свойств, Offset применяется только к объекту Range и ни к какому другому. Данное свойство использует единственный синтаксис.

```
объект.Offset(смещение_строки, смещение_столбца)
```

Два аргумента свойства Offset соответствуют смещению относительно левой верхней ячейки указанного диапазона Range. Эти аргументы могут быть положительными (сдвиг вниз или вправо), отрицательными (вверх или влево) или нулевыми. В приведенном ниже примере значение **12** вводится в ячейку, которая находится под активной.

```
ActiveCell.Offset(1,0).Value = 12
```

В следующем примере значение **15** вводится в ячейку над активной ячейкой:

```
ActiveCell.Offset(-1,0).Value = 15
```

Если активная ячейка находится в строке 1, то свойство Offset в предыдущем примере выдаст ошибку, так как оно возвращает несуществующий объект Range.

Свойство Offset особо эффективно при использовании переменных в цикле (см. следующую главу).

В процессе записи макроса в относительном режиме указания ссылки Excel использует свойство Offset для обращения к ячейкам относительно начальной позиции (т.е. активной в момент начала записи макроса ячейки). Например, для генерации следующего кода применена функция записи макросов. Вначале включим запись макроса (при активной ячейке B1), после введем значение в ячейки B1:B3, а затем вновь вернемся к ячейке B1.

```
Sub Macro1()
    ActiveCell.FormulaR1C1 = "1"
    ActiveCell.Offset(1, 0).Range("A1").Select
    ActiveCell.FormulaR1C1 = "2"
```

```
   ActiveCell.Offset(1, 0).Range("A1").Select
   ActiveCell.FormulaR1C1 = "3"
   ActiveCell.Offset(-2, 0).Range("A1").Select
End Sub
```

При записи макросов используется свойство `FormulaR1C1`. Как правило, для ввода значения в ячейку применяется свойство `Value`. Однако при использовании `FormulaR1C1` или `Formula` результат будет таким же.

Обратите внимание и на то, что полученный код ссылается на ячейку A1. Это довольно странно, так как данная ячейка даже не была задействована в макросе. Такая особенность процедуры записи макросов делает программу даже более сложной, чем необходимо. Можно удалить все ссылки на `Range("A1")`, но макрос все равно будет работать нормально.

```
Sub Modified Macro1()
   ActiveCell.FormulaR1C1 = "1"
   ActiveCell.Offset(1, 0).Select
   ActiveCell.FormulaR1C1 = "2"
   ActiveCell.Offset(1, 0).Select
   ActiveCell.FormulaR1C1 = "3"
   ActiveCell.Offset(-2, 0).Select
End Sub
```

Можно получить еще более эффективную версию макроса (например, ту, которую я написал вручную), который вообще не выполняет выделение ячеек.

```
Sub Macro1()
   ActiveCell = 1
   ActiveCell.Offset(1, 0) = 2
   ActiveCell.Offset(2, 0) = 3
End Sub
```

Что следует знать об объектах

В предыдущих разделах были изложены основы использования объектов (включая коллекции), свойств и методов. Однако при этом мы затронули только верхушку айсберга.

Важные концепции для запоминания

В этом разделе рассматриваются дополнительные концепции, которые незаменимы, если вы собираетесь стать профессиональным программистом на VBA. Эти концепции станут более понятными по мере приобретения опыта работы с VBA, а также после прочтения следующих глав книги.

- **Объекты обладают уникальными свойствами и методами.** Каждый объект имеет собственный набор свойств и методов. Однако некоторые объекты характеризуются общими свойствами (например, `Name`) и методами (например, `Delete`).
- **Для управления объектами их не обязательно выделять.** Это может противоречить обычному представлению об управлении объектами Excel, особенно если вы занимались программированием макросов XLM. Дело в том, что действия над объектами эффективнее выполнять, если их сначала не выделять. При записи макроса в Excel объект обычно сначала выделяется. Это необязательно, хотя существенно замедляет работу макроса.

- **Важно понимать предназначение коллекций.** В большинстве случаев вы будете ссылаться на объект непосредственно, обращаясь к коллекции, к которой он принадлежит. Например, для обращения к объекту `Workbook` с названием `Myfile` необходимо сослаться на коллекцию `Workbooks` следующим образом:

  ```
  Workbooks("Myfile.xlsx")
  ```

  Эта ссылка возвращает объект — рабочую книгу, которая вас интересует.

- **Свойства могут возвращать ссылку на другой объект.** Например, в следующем операторе свойство `Font` возвращает объект `Font`, который содержится в объекте `Range`.

  ```
  Range("A1").Font.Bold = True
  ```

- **Обратиться к одному и тому же объекту можно несколькими способами.** Предположим, что у вас есть рабочая книга с названием `Sales` и это единственная открытая рабочая книга. В ней находится один лист с названием `Summary`. Можете сослаться на этот лист любым из приведенных ниже способов.

  ```
  Workbooks("Sales.xlsx").Worksheets("Summary")
  Workbooks(1).Worksheets(1)
  Workbooks(1).Sheets(1)
  Application.ActiveWorkbook.ActiveSheet
  ActiveWorkbook.ActiveSheet
  ActiveSheet
  ```

  Используемый вами метод обычно зависит от того, насколько хорошо вам известно рабочее пространство. Например, если открыто несколько рабочих книг, то второй или третий метод не подойдет. Если вы будете работать с активным листом (какой бы это ни был лист), то воспользуйтесь последними тремя методами. Абсолютную уверенность в том, что вы ссылаетесь на конкретный лист в конкретной рабочей книге, дает только первый метод.

Узнайте больше об объектах и свойствах

Если это ваше первое знакомство с VBA, то вас могут несколько смутить многочисленные объекты, свойства и методы. Вы можете попытаться получить доступ к свойству, которым объект не обладает, однако в процессе выполнения кода произойдет ошибка. Код будет прерываться в этом месте до тех пор, пока вы не исправите ошибку.

Существует несколько способов получить больше информации об объектах, свойствах и методах.

Прочтите оставшуюся часть книги

Не забывайте, что эта глава является вводной. В остальных главах книги рассмотрены детали и приведены полезные информативные примеры.

Используйте средства записи макросов

Несомненно, наилучший способ ознакомиться с VBA — включить функцию записи макросов и записать отдельные действия, выполненные в Excel. Вы получите информацию о том, какие объекты, свойства и методы относятся к конкретной задаче. Будет лучше, если при записи отображается окно модуля VBA, в котором представлен записываемый код.

Используйте электронную справочную систему

Основной источник подробной информации об объектах, методах и процедурах Excel — электронная справочная система. Многие пользователи забывают об этом ресурсе.

Используйте браузер объектов

Инструмент *Object Browser* (браузер объектов) — это удобное средство, предоставляющее список всех свойств и методов для всех доступных объектов. В VBE окно Object Browser можно отобразить одним из трех способов:

- нажмите клавишу <F2>;
- в строке меню выберите команду View⇨Object Browser (Вид⇨Браузер объектов);
- щелкните на кнопке Object Browser, которая находится на стандартной панели инструментов.

Окно Object Browser показано на рис. 7.15.

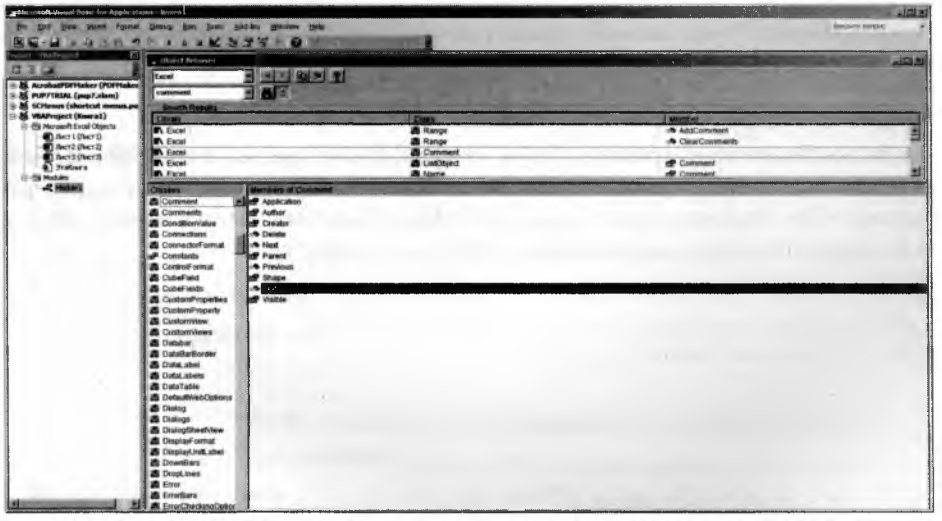


Рис. 7.15. В окне Object Browser можно найти много полезных сведений об объектах

Раскрывающийся список в левом верхнем углу окна Object Browser содержит список всех библиотек объектов, к которым у вас есть доступ:

- собственно Excel;
- MSForms (используется для создания специальных диалоговых окон);
- Office (объекты, общие для всех приложений Microsoft Office);
- Stdole (объекты автоматизации OLE);
- VBA;
- все открытые рабочие книги (каждая книга считается библиотекой объектов, так как содержит объекты).

Ваш выбор в этом списке определяет, что отображается в разделе Classes (Классы), а выбор в разделе Classes обусловливает появление определенных компонентов в поле Members of (Включены в).

После выбора библиотеки можно осуществить поиск конкретной строки текста, чтобы получить список свойств и методов, содержащих данный текст. Это можно сделать, введя текст во втором раскрывающемся списке и щелкнув на значке с изображением бинокля. Предположим, вы работаете над проектом, обрабатывающим примечания в ячейках.

1. Выберите интересующую вас библиотеку. Если вы не знаете, какую именно библиотеку выбрать, укажите вариант <All Libraries>.
2. Введите `Comment` в раскрывающемся списке под списком библиотек.
3. Щелкните на значке в виде бинокля, чтобы начать поиск текста.

В области Search Results (Результаты поиска) отображается текст, соответствующий фрагменту для поиска. Выберите один объект, чтобы отобразить его классы в разделе Classes. Укажите класс, чтобы отобразить его члены (свойства, методы и константы). Обратите внимание на нижнюю часть окна, где приведена дополнительная информация об объекте. Можно нажать <F1>, чтобы перейти непосредственно к необходимому разделу справочной системы.

Структура окна Object Browser может сначала показаться сложной, но, изучив ее, вы убедитесь в ее незаменимости.

Экспериментируйте с окном отладки

Как было отмечено во врезке в одном из предыдущих разделов этой главы, окно отладки (Immediate) в VBE используется для тестирования операторов и проверки разных выражений VBA. Рекомендуется всегда отображать окно отладки на экране, так как оно часто используется для проверки выражений и при отладке кода.

Глава 8

Основы программирования на VBA

В этой главе...

- Обзор элементов и конструкций VBA
- Комментарии
- Переменные, типы данных и константы
- Операторы присваивания
- Массивы
- Объектные переменные
- Пользовательские типы данных
- Встроенные функции
- Управление объектами и коллекциями
- Контроль за выполнением кода

В предыдущих главах вы ознакомились с основами VBA, теперь же пришло время изучить более сложные понятия. В этой главе рассматриваются некоторые ключевые элементы и концепции, связанные с программированием на VBA.

Обзор элементов и конструкций VBA

Если вы ранее использовали другие языки программирования, то большая часть приведенной далее информации покажется вам знакомой. Однако в VBA есть свои тонкости, поэтому даже опытные программисты найдут в данной главе полезную информацию.

В главе 7 речь шла об объектах, свойствах и методах. Но этих знаний еще недостаточно для выполнения сложных задач. Данная глава введет вас в курс дела: в ней анали-

зируются *элементы языка* VBA, такие как ключевые слова и операторы, используемые для написания процедур VBA.

Для начала в качестве примера рассмотрим простую процедуру VBA типа Sub. Она хранится в модуле VBA и вычисляет сумму первых ста положительных целых чисел. По окончании вычислений процедура отображает сообщение с результатом.

```
Sub VBA_Demo()
'  Пример простой процедуры VBA
   Dim Total As Long, i As Long
   Total = 0
   For i = 1 To 100
      Total = Total + i
   Next i
   MsgBox Total
End Sub
```

В этой процедуре применяются некоторые популярные элементы языка, в том числе следующие:

- комментарий (строка, начинающаяся апострофом);
- оператор объявления переменной (строка, начинающаяся ключевым словом Dim);
- две переменные (Total и i);
- два оператора присваивания (Total = 0 и Total = Total + i);
- циклическая структура (For-Next);
- функция VBA (MsgBox).

Все перечисленные элементы языка рассматриваются в следующих разделах главы.

Примечание

Процедуры VBA не всегда управляют объектами. Например, рассмотренная в предыдущем примере процедура работает исключительно с числами.

Ввод кода VBA

Код VBA, находящийся в модуле VBA, состоит из инструкций. Общепринято вводить в каждой строке по одной инструкции. Но это необязательное требование; можно указывать несколько инструкций в одной строке, разделяя их двоеточием. В следующем примере в одной строке находятся четыре инструкции.

```
Sub OneLine()
   x= 1: y= 2: z= 3: MsgBox x + y + z
End Sub
```

Многие программисты полагают, что код воспринимается легче, если в каждой строке находится по одной инструкции.

```
Sub OneLine()
   x = 1
   y = 2
   z = 3
   MsgBox x + y + z
End Sub
```

Строка может быть любой длины; модуль VBA продолжается на следующей строке, когда текущая строка достигает правой границы окна. В длинных строках можно использовать оператор продолжения строки VBA: пробел с подчеркиванием (_).

```
Sub LongLine()
   SummedValue = _
      Worksheets("Лист1").Range("A1").Value + _
      Worksheets("Лист2").Range("A1").Value
End Sub
```

При написании макросов Excel достаточно часто символы подчеркивания применяются для разбиения длинных операторов на несколько строк.

После ввода инструкции редактор VBA выполняет следующие действия для улучшения удобочитаемости кода.

- **Вставляет пробелы между операторами.** Например, если ввести **`Ans=1+2`** (без пробелов), VBA преобразует это выражение следующим образом:

  ```
  Ans = 1 + 2
  ```

- **Изменяет регистр символов ключевых слов, свойств и методов.** Если ввести выражение **`Result=activesheet.range("a1").value=12`**, VBA преобразует его так:

  ```
  Result = ActiveSheet.Range("a1").Value = 12
  ```

  Обратите внимание, что текст в кавычках (в рассматриваемом случае — `"a1"`) не изменился.

- **Поскольку названия переменных VBA нечувствительны к изменению регистра символов, интерпретатор по умолчанию изменяет имена всех переменных, состоящих из букв одного регистра, чтобы их регистр соответствовал последнему введенному варианту.** Например, если сначала переменная определялась как `myvalue` (все символы нижнего регистра), а затем была переопределена в виде `MyValue` (смешанный регистр), VBA изменит название переменной во всех остальных случаях на `MyValue`. Исключение бывает в том случае, когда переменная объявляется с помощью ключевого слова `Dim` либо похожего оператора; название переменной остается неизменным в процессе ее применения.

- **Редактор VBA просматривает инструкции с целью выявления синтаксических ошибок.** После обнаружения ошибки изменяется цвет строки, а на экране появляется сообщение с описанием возникшей проблемы. Воспользуйтесь командой редактора Visual Basic Editor Tools⇨Options (Сервис⇨Параметры) для перехода в диалоговое окно Options (Параметры), в котором можно выбрать цвет выделения ошибки (с помощью вкладки Editor Format (Формат редактора)), а также указать, следует ли отображать сообщение об ошибке (с помощью параметра Auto Syntax Check (Автоматическая проверка синтаксиса) на вкладке Editor (Редактор)).

Комментарии

Комментарий — это описательный текст, который включается в код. VBA полностью игнорирует текст комментария. Комментарии можно использовать произвольно для описаний действий в коде (назначение оператора не всегда очевидно).

Вы можете использовать для комментария новую строку либо вставить комментарий после инструкции в той же строке. Комментарий обозначается апострофом. VBA игнорирует любой текст, введенный после апострофа, за исключением случаев, когда апостроф заключен в кавычки. Например, в следующем операторе комментарий отсутствует, хотя в нем и содержится апостроф:

```
Msg = "Продолжение невозможно"
```

В следующем примере показана процедура VBA с тремя комментариями.

```
Sub Comments()
'   Эта процедура не делает ничего серьезного
    x = 0 'x ничего не содержит
'   Отображение результата
    MsgBox x
End Sub
```

Как правило, в качестве индикатора комментария используется апостроф, однако можно применить для обозначения строки комментария ключевое слово Rem.

```
Rem -- следующий оператор запрашивает имя файла
```

Ключевое слово Rem (сокращение от слова Remark — "пометка") — это, по существу, пережиток старых версий BASIC; его включили в VBA из соображений совместимости. В отличие от апострофа, Rem используется только в начале строки, он не допускается в той же строке, что и инструкция.

Совет

Использование комментариев — удачная идея, но не все комментарии в равной степени предпочтительны. Чтобы комментарий был полезен, он должен содержать информацию, которая неочевидна при рассмотрении самого кода. В противном случае вы просто увеличите общий объем файла приложения.

Ниже приведено несколько общих советов по эффективному использованию комментариев. Итак, обращайтесь к ним:

- для краткого описания назначения каждой созданной процедуры;
- для описания изменений, которые вы вносите в процедуру;
- для указания функции или конструкции, использующейся нестандартным способом;
- для описания назначения переменных (чтобы и вы, и другие пользователи могли истолковать названия переменных, которые без комментария понять сложно);
- для описания способов, которые помогут избежать влияния внутренних ошибок Excel на ваше приложение;
- для вставки комментариев во время написания кода, но не позже.

Совет

В некоторых случаях нужно проверить процедуру, не вставляя в нее инструкцию либо даже целый набор инструкций. Вместо удаления соответствующей инструкции достаточно превратить ее в комментарий путем добавления апострофа в начале строки. После этого VBA проигнорирует инструкцию (или инструкции) при выполнении процедуры. Для преобразования комментария в инструкцию достаточно удалить знак апострофа.

Панель инструментов Edit (Редактирование) в VBE содержит несколько полезных кнопок. Выделите группу инструкций, а затем используйте кнопку Comment Block (Комментировать блок операторов), чтобы преобразовать инструкции в комментарии. Кнопка Uncomment Block (Раскомментировать блок операторов) преобразовывает группу комментариев обратно в инструкции. Указанные кнопки применяются очень часто, поэтому с целью повышения удобства работы можно скопировать их на стандартную (Standard) панель инструментов.

Переменные, типы данных и константы

Главное назначение VBA — обработка данных. Одни данные сохраняются в объектах, например в диапазонах рабочих листов, а другие — в созданных переменных.

Переменная представляет собой именованное место хранения данных в памяти компьютера. Переменные могут содержать *данные разных типов* — от простых логических, или булевых, значений (`True` или `False`) до больших значений с двойной точностью (см. следующий раздел). Значение присваивается переменной с помощью оператора равенства (подробнее об этом — далее). Вы существенно облегчите себе жизнь, если научитесь присваивать переменным простые описательные имена. Однако помните, что VBA поддерживает несколько правил, ограничивающих вас в именовании переменных.

- Вы можете использовать в названиях символы букв, числа и некоторые знаки препинания, но первой в имени переменной всегда должна вводиться буква.
- VBA не различает регистры. Чтобы сделать имена переменных удобочитаемыми, программисты часто используют смешанный регистр (например, `InterestRate`, а не `interestrate`).
- Нельзя использовать в именах пробелы или точки. Чтобы сделать имена переменных более удобными для чтения, программисты вводят символ подчеркивания (`Interest_Rate`).
- Специальные символы объявления типов (`#`, `$`, `%`, `&` или `!`) также не применяются в имени переменной.
- Названия переменных ограничены 254 символами (вряд ли вы, будучи в здравом уме, придумаете настолько длинное название).

В приведенном далее списке содержатся отдельные примеры выражений присваивания, в которых используются различные типы переменных. Названия переменных указываются слева от знака равенства. Каждый оператор присваивает переменной слева от знака равенства значение, которое располагается справа от знака равенства.

```
x = 1
InterestRate = 0.075
LoanPayoffAmount = 243089.87
DataEntered = False
x = x + 1
MyNum = YourNum * 1.25
UserName = "Боб Джонсон"
DateStarted = #12/14/2006#
```

В VBA используется очень много зарезервированных слов, т.е. таких слов, которые не допускается применять в качестве названий переменных или процедур. Если вы попытаетесь ввести одно из таких слов, то будет отображено сообщение об ошибке. Например, несмотря на то что зарезервированное слово `Next` могло бы стать описательным названием многих переменных, следующая инструкция генерирует синтаксическую ошибку:

```
Next = 132
```

К сожалению, сообщения о синтаксических ошибках не всегда достаточно описательны. Указанная выше инструкция генерирует сообщение об ошибке: `Compile error: Expected Variable` (Ошибка компиляции: ожидается переменная). Ситуация не совсем понятна, поэтому при отображении такого сообщения об ошибке обращайтесь к справочной системе, чтобы убедиться в том, что имя переменной не задействовано в VBA в других целях.

Определение типов данных

Интерпретатор VBA облегчает жизнь программистам, автоматически обрабатывая любые типы данных. Не во всех языках программирования управление данными выполняется на столь простом уровне. Например, многие языки программирования являются строго типизированными, т.е. программист должен явно объявлять тип данных каждой используемой переменной.

Тип данных указывает, в каком виде данные хранятся в памяти: как целые значения, действительные числа, текст или др. VBA может автоматически типизировать данные, однако это чревато негативными последствиями — медленным выполнением операций и менее эффективным использованием памяти. В результате, позволяя VBA самостоятельно определять типы данных, вы можете столкнуться с проблемами выполнения больших или сложных приложений. Если вам приходится экономить каждый байт памяти, то вы должны хорошо разбираться в типах данных. Еще одно преимущество явного объявления типа данных всех используемых переменных состоит в том, что хотя VBA дополнительно ищет ошибки на этапе компиляции, подобные ошибки довольно сложно выявить другими способами проверки.

В табл. 8.1 перечислены поддерживаемые в VBA типы данных (специальные типы данных будут описаны далее в этой главе).

Таблица 8.1. Встроенные типы данных VBA

Тип данных	Количество зарезервированных байтов	Диапазон значений
`Byte`	1	0 – 255
`Boolean`	2	`True` (Истина) или `False` (Ложь)
`Integer`	2	–32768 – 32767
`Long`	4	–2147483648 – 2147483647
`Single`	4	–3,402823E38 – –1,401298E-45 (отрицательные числа); 1,401298E–45 – 3,402823E38 (положительные числа)
`Double`	8	–1,79769313486232E308 – –4,94065645841247E–324 (отрицательные числа); 4,94065645841247E–-324 – 1,79769313486232E308 (положительные числа)
`Currency`	8	-922337203685477,5808 – 922337203685477,5807
`Decimal`	12	+/–79228162514264337593543950335 без десятичных знаков; +/–7,9228162514264337593543950335 с 28 знаками после запятой
`Date`	8	1 января 100 года – 31 декабря 9999 года
`Object`	4	Любая ссылка на объект
`String` (переменной длины)	10 + длина строки	0 – приблизительно 2 млрд символов
`String` (фиксированной длины)	Длина строки	1 – приблизительно 65400 символов
`Variant` (числа)	16	Любое числовое значение в пределах диапазона типа данных `Double`. Может также включать специальные значения типа `Empty`, `Error`, `Nothing` и `Null`

Окончание табл. 8.1

Тип данных	Количество зарезервированных байтов	Диапазон значений
`Variant` (символы)	22 + длина строки	0 – примерно 2 млрд
Пользовательский	Зависит от типа	Зависит от элемента

Примечание

Тип данных `Decimal` необычен тем, что его нельзя объявить. На самом деле он является подтипом типа `Variant`. Для преобразования в десятичный тип данных типа `Variant` воспользуйтесь функцией VBA `CDec`.

Рекомендуется выбирать тот тип данных, в котором используется минимальное количество байтов для хранения значений, но он также должен быть достаточным для представления максимальных значений переменных. При работе с данными в VBA скорость выполнения операций зависит от объема данных, которые обрабатываются с помощью VBA. Другими словами, чем меньше байтов зарезервировано для данных, тем быстрее VBA получает доступ к данным и обрабатывает их.

Для проведения математических вычислений в рабочих листах Excel использует тип данных `Double`. Его рекомендуется применять и в процессе обработки чисел в VBA для обеспечения той же точности вычислений. Для обработки целочисленных значений идеально подходит тип `Integer`, если, конечно, вы уверены, что используемые значения не превышают 32767. В противном случае обратитесь к типу данных `Long`. При управлении номерами строк в рабочем листе Excel лучше применять тип данных `Long`, так как количество строк в рабочем листе превышает максимальное значение, допустимое в типе данных `Integer`.

Тестирование типов данных

Чтобы оценить важность определения типа данных, автор разработал специальную процедуру, выполняющую циклические вычисления с последующим отображением затраченного времени.

```
Sub TimeTest()
   Dim x As Long, y As Long
   Dim A As Double, B As Double, C As Double
   Dim i As Long, j As Long
   Dim StartTime As Date, EndTime As Date
'  Сохранение времени начала вычислений
   StartTime = Timer
'  Выполнение вычислений
   x = 0
   y = 0
   For i = 1 To 10000
      x = x + 1
      y = x + 1
      For j = 1 To 10000
         A = x + y + i
         B = y - x - i
         C = x / y * i
      Next j
   Next i
'  Получение времени окончания вычислений
```

```
    EndTime = Timer
'   Отображение общего времени в секундах
    MsgBox Format(EndTime - StartTime, "0.0")
End Sub
```

На компьютере автора выполнение этой процедуры заняло 7,7 секунды (этот показатель определяется вашей системой). Затем были закомментированы операторы `Dim`, с помощью которых объявляются типы данных. Для этого в начале каждой строки, содержащей операторы `Dim`, были вставлены апострофы. В результате интерпретатор VBA использовал тип данных, заданный по умолчанию, а именно — `Variant`. После этого процедура была выполнена еще раз.

Мораль проста: для скорейшего выполнения VBA-приложений объявляйте переменные!

Рабочая книга, содержащая этот код, находится на прилагаемом к книге компакт-диске в файле `timing text.xlsm`.

Объявление переменных

Если вы не объявили тип данных для переменной, используемой в процедуре VBA, по умолчанию будет задан тип данных `Variant`. Данные с типом `Variant` имеют только одно заслуживающее внимания свойство: они изменяют свой тип в зависимости от того, какие операции над ними выполняются.

В следующей процедуре показано, каким образом переменная принимает разные типы данных.

```
Sub VariantDemo()
    MyVar = "123"
    MyVar = MyVar / 2
    MyVar = "Ответ: " & MyVar
    MsgBox MyVar
End Sub
```

В процедуре `VariantDemo` переменная `MyVar` вначале выступает строкой из трех символов. Затем эта "строка" делится на два и приобретает числовой тип данных. После этого `MyVar` присоединяется к строке, что вызывает обратное преобразование `MyVar` в строку. Оператор `MsgBox` отображает окончательное строковое значение: `Ответ: 61,5`.

Чтобы проиллюстрировать проблемы, которые могут возникнуть при обработке типа данных `Variant`, рассмотрим следующую процедуру.

```
Sub VariantDemo2()
    MyVar = "123"
    MyVar = MyVar + MyVar
    MyVar = "Ответ: " & MyVar
    MsgBox MyVar
End Sub
```

При выполнении этой процедуры в окне сообщений появляется сообщение `Ответ: 123123`. Скорее всего, это не тот результат, которого вы ожидали. В процессе управления текстовыми данными, представленными типом `Variant`, оператор "+" выполняет конкатенацию строк.

Определение типа данных

Для определения типа данных переменной используется функция VBA `TypeName`. Ниже представлена модифицированная версия предыдущей процедуры. Эта процедура на каждом шаге отображает тип данных переменной `MyVar`. Вы увидите, что сначала

переменная является строкой, затем преобразуется в числовой тип Double и в завершение снова становится строкой.

```
Sub VariantDemo2()
   MyVar = "123"
   MsgBox TypeName(MyVar)
   MyVar = MyVar / 2
   MsgBox TypeName(MyVar)
   MyVar = "Ответ: " & MyVar
   MsgBox TypeName(MyVar)
   MsgBox MyVar
End Sub
```

Благодаря VBA преобразование типов для необъявленных переменных выполняется автоматически. Этот процесс может показаться удачным выходом из создавшейся ситуации, однако помните, что при этом снижается скорость обработки данных, быстрее заполняется свободная память, а также повышается риск возникновения ошибок.

Объявлять каждую переменную в процедуре перед ее использованием — замечательная привычка. В процессе объявления переменной вы явно указываете ее название и тип данных. Объявление переменных дает два основных преимущества.

- **Программы работают быстрее и используют память более эффективно.** Тип данных по умолчанию Variant резервирует больше памяти, чем необходимо, и вызывает многократную проверку данных, занимающую процессорное время. Если программа точно знает тип данных, она не выполняет дополнительную проверку данных и резервирует ровно столько памяти, сколько необходимо для хранения конечных данных.
- **Объявив переменные, можно избежать ошибок, связанных с неправильным введением имен переменных.** Предполагается, что вы используете оператор Option Explicit, делающий обязательным объявление всех переменных (см. следующий раздел). Предположим, что вы также в коде применяете необъявленную переменную с названием CurrentRate. Наряду с этим на определенном этапе процедуры вы обращаетесь к оператору CurentRate = 0.75. Такая ошибка в названии переменной, которую тяжело выявить, скорее всего, будет причиной неправильных результатов.

Обязательное объявление всех переменных

Чтобы обеспечить обязательное объявление всех используемых переменных, необходимо включить следующую строку в качестве первой инструкции в модуле VBA:

```
Option Explicit
```

Этот оператор отвечает за то, что программа будет приостанавливаться всякий раз при нахождении в ней необъявленной переменной. VBA выведет сообщение об ошибке (рис. 8.1); для продолжения процедуры следует объявить переменную.

О примерах, приведенных в этой главе

Настоящая глава содержит ряд примеров кода VBA, обычно представленных в виде простых процедур. Все примеры призваны объяснять наиболее простым образом различные концепции программирования. По этой причине в большинстве случаев поставленные задачи выполняются неэффективно. Другими словами, не следует с точностью использовать эти примеры в своих проектах. В следующих главах книги приведенные примеры представляют действительную ценность.

Рис. 8.1. Интерпретатор VBA сообщает о том, что процедура использует необъявленную переменную

Совет

Чтобы оператор `Option Explicit` автоматически включался при вставке нового модуля VBA, выберите параметр Require Variable Declaration (Обязательное объявление переменных) на вкладке Editor (Редактор) в диалоговом окне VBE Options (Параметры редактора Visual Basic). Для этого воспользуйтесь командой Tools⇨Options (Сервис⇨Параметры). Настоятельно рекомендуем вам воспользоваться этой установкой, хотя она не оказывает влияния на существующие модули.

Область действия переменной

Область действия переменной определяет, в каких модулях и процедурах она может использоваться. Существуют следующие типы областей действия переменных. В табл. 8.2 перечислены три области действия переменной.

Таблица 8.2. Области действия переменных

Область действия	Способ объявления переменных
Отдельная процедура	В процедуру включается оператор `Dim` или `Static`
Отдельный модуль	Перед первой процедурой в модуле включается оператор `Dim` или `Private`
Все модули	Перед первой процедурой в модуле указывается оператор `Public`

Все типы областей действия рассматриваются в следующих разделах.

Локальные переменные

Локальная переменная — это переменная, объявленная в процедуре. Локальные переменные могут использоваться только в процедуре, в которой они объявлены. После

выполнения процедуры переменная становится невостребованной, поэтому Excel освобождает соответствующую область памяти. Если нужно сохранить значение переменной, объявите ее как `Static` (см. раздел "Переменные `Static`").

Наиболее популярный способ объявить локальную переменную — вставить оператор `Dim` между операторами `Sub` и `End Sub`. Операторы `Dim` обычно вводятся непосредственно после оператора `Sub`, перед кодом процедуры.

Примечание

Если вас интересует происхождение ключевого слова `Dim`, то примите к сведению, что это сокращение от "Dimension" — "размерность". В прежних версиях BASIC этот оператор использовался исключительно для объявления размерности массива. В VBA ключевое слово `Dim` применяется для объявления любой переменной, а не только массивов.

В представленной далее процедуре используется шесть локальных переменных, объявленных с помощью операторов `Dim`.

```
Sub MySub()
   Dim x As Integer
   Dim First As Long
   Dim InterestRate As Single
   Dim TodaysDate As Date
   Dim UserName As String
   Dim MyValue
'  - [Здесь указывается код процедуры] -
End Sub
```

Обратите внимание, что последний оператор `Dim` в этом примере объявляет не тип данных, а только саму переменную. В результате переменная приобретает тип `Variant`.

Кроме того, можно объявить несколько переменных, воспользовавшись единственным оператором `Dim`.

```
Dim x As Integer, y As Integer, z As Integer
Dim First As Long, Last As Double
```

Предупреждение

В отличие от других языков программирования, в VBA нельзя объявить тип данных одновременно для группы переменных, разделив переменные запятыми. Например, следующий оператор является корректным, хотя и не объявляет переменные как `Integer`:

```
Dim i, j, k As Integer
```

В VBA только `k` объявляется как `Integer`; остальные переменные объявляются как `Variant`. Для объявления переменных `i`, `j` и `k` в виде `Integer` используйте следующий оператор:

```
Dim i As Integer, j As Integer, k As Integer
```

Если переменная объявлена как локальная, другие процедуры в том же модуле могут использовать подобное имя, но каждый экземпляр переменной считается уникальным в своей процедуре.

Как правило, локальные переменные — самые эффективные, так как VBA освобождает память, которую они используют, после окончания процедуры.

Еще один способ указания типов данных для переменных

Как и в большинстве других версий BASIC, язык VBA позволяет присоединить символ к названию переменной, чтобы указать ее тип данных. Например, можно объявить переменную `MyVar` как целое число, добавив к ее названию символ `%`.

```
Dim MyVar%
```

Символы объявления типов данных представлены для большинства типов данных VBA. Отсутствующие в таблице типы данных не имеют собственного символа объявления типа.

Тип данных	Символ объявления типа
Integer	%
Long	&
Single	!
Double	#
Currency	@
String	$

Этот метод типизации данных — пережиток старых версий BASIC; лучше объявлять переменные с помощью других методов, описанных в этой главе. Символы объявления типов перечислены здесь только потому, что вы можете встретиться с ними в старых программах.

Переменные уровня модуля

Иногда необходимо, чтобы переменная была доступна во всех процедурах модуля. В таком случае объявите переменную *перед* первой процедурой модуля (за пределами процедур или функций).

В приведенном ниже примере оператор `Dim` — первая инструкция в модуле. Обе процедуры, `Procedure1` и `Procedure2`, имеют доступ к переменной `CurrentValue`.

```
Dim CurrentValue as Integer
Sub Procedure1()
'    - [Здесь вводится код процедуры] -
End Sub
Sub Procedure2()
'    - [Здесь вводится код процедуры] -
End Sub
```

Обычно значение переменной уровня модуля не изменяется к окончанию выполнения процедуры (по достижении оператора `End Sub` либо `End Function`). Исключение из этого правила бывает в случае применения оператора `End`. Как только интерпретатор VBA выполняет оператор `End`, все переменные уровня модуля теряют свои значения.

Переменные `Public`

Чтобы сделать переменную доступной во всех процедурах всех модулей VBA проекта, необходимо объявить переменную на уровне модуля (перед объявлением первой процедуры) с помощью ключевого слова `Public`, а не `Dim`.

```
Public CurrentRate as Long
```

Ключевое слово `Public` делает переменную `CurrentRate` доступной для любой процедуры проекта, даже для процедур, которые располагаются в других модулях проекта. Этот оператор следует вставить перед первой процедурой модуля. Более того, подоб-

ный код объявления переменных должен вводиться в стандартном модуле VBA, а не в коде модуля листа или формы.

Переменные Static

Переменные Static — особый случай. Они объявляются на уровне процедуры и сохраняют свои значения после нормального завершения процедуры. Но если выполнение процедуры прерывается с помощью оператора End, статические переменные теряют свои значения.

Объявление статических переменных осуществляется с помощью ключевого слова Static.

```
Sub MySub()
   Static Counter as Integer
   ' - [Здесь находится код процедуры] -
End Sub
```

Правила именования переменных

Некоторые программисты называют переменные так, что типы данных легко определить по одному только названию. Однако такой прием усложняет восприятие кода.

Распространенное правило именования связано с использованием стандартной приставки в нижнем регистре в названии переменной. Например, булеву переменную, которая контролирует сохранение рабочей книги, можно назвать bWasSaved. Это переменная типа Boolean. В следующей таблице перечислены стандартные префиксы для некоторых типов данных.

Тип данных	Префикс
Boolean	b
Integer	i
Long	l
Single	s
Double	d
Currency	c
Date/Time	dt
String	str
Object	obj
Variant	v
Пользовательский	u

Работа с константами

Значение переменной может изменяться при выполнении процедуры (часто так и случается, поэтому она и называется "переменной"). Иногда же необходимо использовать именованное значение или строку, которая никогда не изменяется, — константу.

Использование констант в программе вместо строго запрограммированных значений или строк — удачный прием программирования. Например, если процедура несколько раз ссылается на определенное значение, например на процентную ставку, следует объявить это значение как константу и использовать в выражениях имя константы, а не ее значение. Такой прием не только делает программу более удобной для восприятия, но и облегчает последующее изменение кода — достаточно изменить только одну инструкцию, а не несколько.

Объявление констант

Для объявления констант используется оператор Const. Ниже приведено несколько примеров.

```
Const NumQuarters as Integer = 4
Const Rate = .0725, Period = 12
Const ModName as String = "Budget Macros"
Public Const AppName as String = "Budget Application"
```

Во втором примере тип данных не объявлен. Следовательно, VBA определяет тип данных на основе их значений. Переменная Rate имеет тип данных Double, а переменная Period — Integer. Так как константы никогда не изменяют своего значения, обычно их объявляют в виде конкретного типа данных.

Как и переменные, константы имеют область действия. Если требуется, чтобы константа была доступна только в одной процедуре, объявите ее после оператора Sub или Function — и она станет локальной. Вы сделаете константу доступной для всех процедур в модуле, если объявите ее перед первой процедурой модуля. Чтобы сделать константу доступной для всех модулей рабочей книги, используйте ключевое слово Public и объявите константу перед первой процедурой модуля.

```
Public Const InterestRate As Double = 0.0725
```

Примечание

При попытке изменения значения константы в коде VBA вы получите сообщение об ошибке (чего и следовало ожидать). Константа — это постоянное значение, а не переменная.

Использование предопределенных констант

В Excel и VBA существует целый ряд предопределенных констант, которые можно использовать без объявления. Вам даже необязательно знать значение этих констант для их применения. При записи макросов обычно используются константы, а не значения. В следующей процедуре для изменения ориентации страницы активного листа на альбомную применена встроенная константа (xlLandscape).

```
Sub SetToLandscape()
   ActiveSheet.PageSetup.Orientation = xlLandscape
End Sub
```

Справочные сведения о константе xlLandscape можно найти в справочной системе. Кроме того, если у вас включен параметр AutoList Members (Автоматическая вставка объектов), то вы можете получить помощь непосредственно при вводе кода (рис. 8.2). Во многих случаях VBA автоматически перечисляет все константы, присваиваемые определенному свойству.

Фактическое значение переменной xlLandscape равно 2. Еще одна встроенная константа для изменения ориентации страницы — xlPortrait — имеет значение 1. Очевидно, что при использовании встроенных констант необязательно знать их значения.

Примечание

Окно Object Browser, которое будет рассмотрено в главе 7, содержит список всех констант Excel и VBA. Чтобы открыть Object Browser в VBE, нажмите клавишу <F2>.

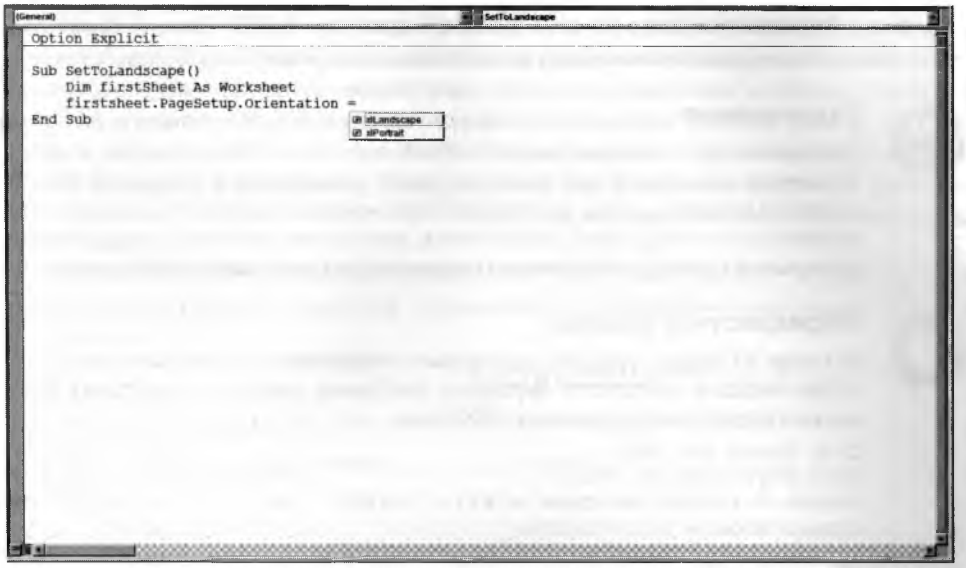


Рис. 8.2. VBA отображает список констант, которые были назначены свойству

Управление строками

Как и Excel, VBA может работать не только с числами, но и с текстом (строками). В VBA представлено два типа строк.

- *Строки фиксированной длины* объявляются с определенным количеством символов. Максимальная длина строки — 65535 символов.
- *Строки переменной длины* теоретически могут вмещать до 2 млрд символов.

Каждый символ в строке занимает 1 байт памяти, к тому же память дополнительно используется для хранения заголовка строки. При объявлении строки с помощью оператора `Dim` вы можете определить ее длину, если она вам известна (задать тип строки с фиксированной длиной), или активизировать динамическую обработку длины строки (задать тип строки переменной длины). Работа со строками фиксированной длины несколько эффективнее с точки зрения использования памяти.

В следующем примере переменная `MyString` объявляется как строка с максимальной длиной 50 символов. `YourString` также объявлена как строка, но имеет переменную длину.

```
Dim MyString As String * 50
Dim YourString As String
```

Работа с датами

Конечно, для хранения даты можно использовать строку, но над ней нельзя выполнять вычисления. Применение специального типа данных `Date` — наилучший способ управления датами.

Переменная, определенная как `Date`, занимает 8 байт памяти и может содержать даты от 1 января 100 года до 31 декабря 9999 года. Этого диапазона почти в 10000 лет более чем достаточно даже для самого невероятного финансового прогноза! Тип данных

Date также применяется для хранения значений времени. В VBA дата и время определяются как значения, заключенные между знаками # (см. далее).

Примечание

Диапазон дат, которые можно обрабатывать в VBA, намного шире собственного диапазона дат Excel, который начинается с 1 января 1900 года. Поэтому следите, чтобы в рабочем листе не использовались данные, которые находятся за пределами допустимого диапазона дат Excel. Ниже приведены отдельные примеры объявления переменных и констант с типом данных Date.

Перекрестная ссылка

В главе 10 будут описаны некоторые относительно простые функции VBA, позволяющие создавать формулы рабочего листа, с помощью которых можно обрабатывать даты до 1900 года.

```
Dim Today As Date
Dim StartTime As Date
Const FirstDay As Date = #1/1/2007#
Const Noon = #12:00:00#
```

Предупреждение

Даты всегда определяются в формате "месяц/день/год", даже если система настроена на отображение данных в другом формате (например, "день/месяц/год").

Если вы используете для отображения даты окно сообщений, дата будет представлена в соответствии с коротким форматом даты, установленным в системе. Аналогично время отображается согласно системному формату (12- или 24-часовому). Вы можете модифицировать системные настройки с помощью аплета Язык и региональные стандарты панели управления в Windows.

Ошибка дат в Excel

В Excel (это общеизвестный недостаток) имеется ошибка, связанная с датами. Она основана на том, что 1900 год — високосный. Поэтому в Excel можно представить дату 29 февраля 1900 года, хотя на самом деле такой даты не существует.

```
=Date(1900,2,29)
```

В VBA не существует подобной ошибки. Эквивалентом функции Excel DATA в VBA является функция DateSerial. Представленное ниже выражение (корректно) возвращает 1 марта 1900 года.

```
DateSerial(1900,2,29)
```

Следовательно, система представления дат в Excel не соответствует системе представления дат в VBA. Эти две системы возвращают различные значения для дат в диапазоне от 1 января 1900 года до 28 февраля 1900 года.

Операторы присваивания

Оператор присваивания — это инструкция VBA, выполняющая математическое вычисление и присваивающая результат переменной или объекту. В справочной системе Excel выражение определяется как комбинация ключевых слов, операторов, переменных

и констант. Эта комбинация возвращает в результате строку, число или объект. Выражение может выполнять вычисление, обрабатывать символы или тестировать данные.

Нельзя придумать лучшего определения. Большое количество операций, выполняемых в VBA, связано с разработкой (и отладкой) выражений. Если вы знаете, как создавать формулы в Excel, то у вас не будет возникать проблем с созданием выражений в VBA. В формуле рабочего листа Excel результат отображается в ячейке. С другой стороны, выражение VBA может присваивать значение переменной или использоваться как значение свойства.

В VBA оператором присваивания выступает знак равенства (=). Ниже приведены примеры использования операторов присваивания (выражения приводятся справа от знака равенства).

```
x = 1
x = x + 1
x = (y * 2) / (z * 2)
FileOpen = True
FileOpen = Not FileOpen
Range("TheYear").Value = 2010
```

Совет

Выражения могут быть очень сложными. Чтобы сделать длинные выражения более удобными для восприятия, используйте символ продолжения строки (пробел с подчеркиванием).

Зачастую в выражениях применяются функции. Это могут быть встроенные функции VBA, функции рабочих листов Excel или специальные функции, разработанные в VBA. Подробнее встроенные функции VBA рассматриваются далее в главе.

В VBA математические операторы играют главную роль. Известные вам операторы описывают математические операции, в том числе сложение (+), умножение (*), деление (/), вычитание (-), возведение в степень (^) и конкатенацию строк (&). Менее известные операторы — обратная косая черта (\) (используется в целочисленном делении) и оператор Mod (применяется при определении модуля числа). Оператор Mod возвращает остаток от деления одного числа на другое. Например, следующее выражение возвращает 2:

```
17 Mod 3
```

VBA поддерживает операторы сравнения, которые применяются в формулах Excel: равно (=), больше (>), меньше (<), больше или равно (>=), меньше или равно (<=) и не равно (<>).

Предупреждение

Оператор отрицания (знак "минус") в Excel и VBA интерпретируется по-разному. При работе в Excel следующая формула возвращает значение 25.

```
=-5^2
```

После выполнения следующего кода VBA переменная x получит значение -25.

```
x = -5 ^ 2
```

Полученный результат объясняется тем, что в VBA сначала выполняется операция возведения в степень, а затем применяется оператор отрицания. А вот следующий код вернет нужное нам значение 25.

```
x = (-5) ^ 2
```

За единственным исключением порядок приоритетности выполнения операторов в VBA совпадает с очередностью выполнения операторов в Excel (табл. 8.3). Если нужно ее изменить, используйте скобки.

Таблица 8.3. Приоритетность выполнения операторов

Оператор	Выполняемая операция	Приоритетность оператора
^	Возведение в степень	1
* и /	Умножение и деление	2
+ и -	Сложение и вычитание	3
&	Конкатенация	4
=, <, >, <=, >=, <>	Сравнение	5

В результате выполнения следующего оператора переменная x получает значение 10, поскольку оператор умножения имеет больший приоритет, чем оператор сложения.

```
x = 4 + 3 * 2
```

Во избежание неоднозначности можно записать оператор следующим образом:

```
x = 4 + (3 * 2)
```

В VBA также имеется полный набор логических операторов, которые показаны в табл. 8.4. Дополнительные сведения об этих операторах (вместе с примерами) можно найти в справочной системе VBA.

Таблица 8.4. Логические операторы VBA

Оператор	Действие
Not	Логическое отрицание выражения
And	Логическая конъюнкция двух выражений
Or	Логическая дизъюнкция двух выражений
Xor	Логическое исключение двух выражений
Eqv	Логическая эквивалентность двух выражений
Imp	Логическая импликация двух выражений

В следующей инструкции используется оператор Not для отображения линий сетки в активном окне. Свойство DisplayGridLines принимает значение True или False. Следовательно, оператор Not изменяет True на False, а False — на True.

```
ActiveWindow.DisplayGridlines = _
   Not ActiveWindow.DisplayGridlines
```

Представленное далее выражение осуществляет логическую операцию And. Функция MsgBox отображает значение True, только если Лист1 — активный лист *и* активная ячейка находится в строке 1. Если одно или оба этих условия неверны, функция MsgBox отображает значение False.

```
MsgBox ActiveSheet.Name = "Лист1" And ActiveCell.Row = 1
```

Следующее выражение осуществляет логическую операцию Or. Оператор MsgBox отображает True, если активен Лист1 или Лист2.

```
MsgBox ActiveSheet.Name = "Лист1" _
   Or ActiveSheet.Name = "Лист2"
```

Массивы

Массив — это группа элементов одного типа, которые имеют общее имя; на конкретный элемент массива ссылаются, используя имя массива и индекс. Например, можно определить массив из 12 строк так, чтобы каждая переменная соответствовала названию месяца. Если вы назовете массив `MonthNames`, то можете обратиться к первому элементу массива как `MonthNames(0)`, ко второму — как `MonthNames(1)` и так до `MonthNames(11)`.

Объявление массивов

Массив, как и обычные переменные, объявляется с помощью операторов `Dim` и `Public`. Кроме того, можно определить количество элементов в массиве: введите нижний индекс, ключевое слово `To` и верхний индекс — и вся конструкция будет заключена в скобки. Например, объявить массив, содержащий ровно 100 целых чисел, можно следующим образом:

```
Dim MyArray(1 To 100) As Integer
```

Совет

При объявлении массива в обязательном порядке указывается только верхний индекс; тогда VBA установит нижний индекс равным 0. Следовательно, два следующих оператора приведут к одинаковым результатам.

```
Dim MyArray(0 to 100) As Integer
Dim MyArray(100) As Integer
```

В обоих случаях массив состоит из 101 элемента.

По умолчанию в массивах VBA в качестве первого элемента используется нуль. Если вы хотите, чтобы в качестве первого индекса всех массивов использовалась единица, то перед первой процедурой модуля нужно сделать следующее объявление:

```
Option Base 1
```

Объявление многомерных массивов

В примерах массивов в предыдущем разделе использовались одномерные массивы. Массивы VBA могут иметь до 60 измерений, хотя на самом деле используется не более трех (трехмерные массивы). Показанный ниже оператор объявляет двухмерный 100-элементный массив целых чисел.

```
Dim MyArray(1 To 10, 1 To 10) As Integer
```

Этот массив можно рассматривать как матрицу значений размером 10×10. Чтобы обратиться к конкретному элементу двухмерного массива, используйте два индекса. Например, таким образом присваивается значение элементу предыдущего массива.

```
MyArray(3, 4) = 125
```

Трехмерный массив, содержащий 1000 элементов, можно представлять в виде куба.

```
Dim MyArray(1 To 10, 1 To 10, 1 To 10) As Integer
```

Чтобы обратиться к элементу внутри массива, используйте три индекса.

```
MyArray(4, 8, 2) = 0
```

Объявление динамических массивов

Динамический массив не имеет предопределенного количества элементов. Он объявляется с незаполненными значениями в скобках.

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

Тем не менее, прежде чем динамический массив можно будет использовать в программе, необходимо обратиться к оператору ReDim, указывающему VBA, сколько элементов находится в массиве. Для этого часто применяется переменная, значение которой неизвестно до тех пор, пока процедура не будет запущена на выполнение. Например, если переменной x присвоено число, размер массива определяется с помощью следующего оператора:

```
ReDim MyArray (1 to x)
```

Оператор ReDim можно использовать повторно, изменяя размер массива по мере необходимости. При изменении количества размерностей массива имеющиеся значения утрачиваются. Если нужно сохранить эти значения, воспользуйтесь оператором ReDim Preserve.

```
ReDim Preserve MyArray (1 to y)
```

Подробнее массивы будут рассмотрены в разделе "Циклическая обработка инструкций".

Объектные переменные

Объектая переменная — это переменная, представляющая целый объект, например диапазон или рабочий лист. Объектные переменные имеют особое значение по следующим причинам:

- они значительно упрощают программу;
- они ускоряют выполнение программы.

Объектные переменные, как и обычные переменные, объявляются с помощью оператора Dim или Public. Например, в следующем операторе переменная InputArea объявляется как объект Range:

```
Dim InputArea As Range
```

Для присваивания объекта переменной воспользуйтесь ключевым словом Set:

```
Set InputArea = Range("C16:E16")
```

Чтобы увидеть, каким образом объектные переменные упрощают код, обратите внимание на следующую процедуру, в которой не используется объектная переменная.

```
Sub NoObjVar()
   Worksheets("Лист1").Range("A1").Value = 124
   Worksheets("Лист1").Range("A1").Font.Bold = True
   Worksheets("Лист1").Range("A1").Font.Italic = True
   Worksheets("Лист1").Range("A1").Font.Size = 14
   Worksheets("Лист1").Range("A1").Font.Name = "Cambria"
End Sub
```

Эта процедура вводит значение в ячейку A1 листа Лист1 активной рабочей книги, а затем применяет некоторое форматирование, изменяя шрифт и размер. В примере введено много кода для решения такой простой задачи. Чтобы упростить ее, сведите процедуру к использованию объектной переменной.

```
Sub ObjVar()
   Dim MyCell As Range
```

```
   Set MyCell = Worksheets("Лист1").Range("A1")
   MyCell.Value = 124
   MyCell.Font.Bold = True
   MyCell.Font.Italic = True
   MyCell.Font.Size = 14
   MyCell.Font.Name = Cambria
End Sub
```

После объявления переменной `MyCell` как объекта `Range` оператор `Set` присваивает ей сам объект. В результате в следующих операторах используется упрощенная ссылка `MyCell` вместо длинной `Worksheets("Лист1").Range("A1")`.

Совет

После присвоения объекта переменной VBA получает доступ к нему намного быстрее, чем с помощью непосредственной ссылки на объект. Поэтому, если скорость имеет значение, используйте объектные переменные. Удобно рассматривать этот феномен в категориях узлов. Каждый раз, когда VBA встречает новый узел, например `Sheets(1).Range("AI")`, ему требуется определенное время на расшифровку ссылки. Чем меньше узлов указано в ссылке, тем быстрее обрабатывается последняя. Другой способ увеличить скорость выполнения программы — воспользоваться конструкцией `With-End With`, которая уменьшает количество обрабатываемых узлов. Подробнее эта конструкция будет рассмотрена ниже.

Настоящая ценность объектных переменных станет вам понятной после рассмотрения циклов.

Пользовательские типы данных

VBA позволяет создавать специальные, или пользовательские, типы данных (эта концепция напоминает записи Паскаля или структуры C). Определенный пользователем тип данных может облегчить управление некоторыми типами данных. Например, если приложение обрабатывает сведения о клиенте, то можно создать пользовательский тип данных с названием `CustomerInfo`.

```
Type CustomerInfo
   Company As String
   Contact As String
   RegionCode As Long
   Sales As Double
End Type
```

Примечание

Пользовательские типы данных определяются в верхней части модуля перед началом процедур.

Если пользовательский тип данных уже создан, для объявления переменной этого типа примените оператор `Dim`. Обычно пользовательский тип данных определяется для массивов.

```
Dim Customers(1 To 100) As CustomerInfo
```

Все 100 элементов этого массива состоят из четырех компонентов (как указано в пользовательском типе данных — `CustomerInfo`). Вы можете сослаться на конкретный компонент элемента следующим образом.

```
Customers(1).Company = "Acme Tools"
Customers(1).Contact = "Tim Robertson"
Customers(1).RegionCode = 3
Customers(1).Sales = 150674.98
```

Вам также предоставлена возможность управлять элементом массива как одним целым. Например, чтобы скопировать информацию из `Customers(1)` в `Customers(2)`, используется следующая инструкция:

```
Customers(2) = Customers(1)
```

Предыдущий пример эквивалентен приведенному ниже блоку инструкций.

```
Customers(2).Company = Customers(1).Company
Customers(2).Contact = Customers(1).Contact
Customers(2).RegionCode = Customers(1).RegionCode
Customers(2).Sales = Customers(1).Sales
```

Встроенные функции

Как и в большинстве других языков программирования, в VBA есть ряд встроенных функций, упрощающих вычисления и операции. Часто функции позволяют выполнить операции, которые по-другому осуществить сложно или вообще невозможно. Многие функции VBA подобны (или идентичны) функциям Excel. Например, функция VBA `UCase`, преобразующая строку в верхний регистр, эквивалентна функции Excel ПРОПИСН.

Перекрестная ссылка

В приложении Б находится полный список функций VBA, включающий краткое описание каждой из них. Все эти функции описаны в справочной системе VBA.

Совет

Чтобы получить список функций VBA при написании кода, введите VBA и точку. После этого VBE отобразит список всех вложенных в объект VBA объектов, включая функции (рис. 8.3). Функции обозначаются зеленым значком. Если этот способ вам недоступен, проверьте, включен ли параметр Auto List Members (Автоматическая вставка объектов). Для этого выберите команду Tools⇨Options (Сервис⇨Параметры) и щелкните на вкладке Editor (Редактор).

Функции VBA используются в выражениях почти так же, как и функции Excel в формулах рабочего листа. Например, можно создавать вложенные функции VBA. Ниже приведена простая процедура, которая вычисляет квадратный корень переменной с помощью функции `Sqr` VBA, сохраняет результат в другой переменной и затем отображает результат.

```
Sub ShowRoot()
   Dim MyValue As Double
   Dim SquareRoot As Double
   MyValue = 25
   SquareRoot = Sqr(MyValue)
   MsgBox SquareRoot
End Sub
```

Рис. 8.3. Отображение списка функций VBA в среде VBE

Функция `Sqr` VBA эквивалентна функции рабочего листа КОРЕНЬ в Excel.

Вы можете использовать ряд (но не все) функций Excel в коде VBA. Объект `WorksheetFunction`, который содержится в объекте `Application`, располагает всеми функциями рабочего листа, которые можно вызвать в процедуре VBA.

Чтобы использовать функцию Excel в операторе VBA, перед названием функции введите следующее выражение:

```
Application.WorksheetFunction
```

Функция MsgBox

Функция `MsgBox` — одна из самых полезных в VBA. Во многих примерах из этой главы она используется для отображения значений переменных.

Данная функция часто является достойной заменой простой форме. Кроме того, это превосходный инструмент отладки, поскольку вы можете в любое время вставить функцию `MsgBox`, чтобы приостановить программу и отобразить результат вычисления или присваивания.

Как правило, функции возвращают одно значение, которое можно присвоить переменной. Функция `MsgBox` не только возвращает значение, но и отображает диалоговое окно, в котором пользователь может выполнить определенные действия. Значение, возвращаемое функцией `MsgBox`, является ответом пользователя на отображенный запрос. Функция `MsgBox` может применяться даже в том случае, когда ответ пользователя не требуется, а нужно отобразить сообщение.

Формальный синтаксис функции `MsgBox` предполагает использование пяти аргументов (аргументы в квадратных скобках — необязательные).

`MsgBox(`*`сообщение`*`[, `*`кнопки`*`] [, `*`заголовок`*`] [, `*`файл_справки`*`, `*`контекст`*`])`

- *`Сообщение`* (обязательный аргумент) — сообщение, которое отображается в диалоговом окне.

- *Кнопки* (необязательный аргумент) — значение, определяющее, какие кнопки и пиктограммы (если нужно) отображаются в окне сообщения. Применяйте встроенные константы (например, vbYesNo).
- *Заголовок* (необязательный аргумент) — текст, который отображается в строке заголовка окна сообщения. По умолчанию отображается текст Microsoft Excel.
- *Файл_справки* (необязательный аргумент) — название файла справки, соответствующего окну сообщения.
- *Контекст* (необязательный аргумент) — контекстный идентификатор раздела справки. Представляет конкретный раздел справки для отображения. Если используется аргумент *контекст*, следует также задействовать аргумент *файл_справки*.

Вы можете присвоить полученное значение переменной либо использовать функцию без оператора присваивания. В приведенном ниже примере результат присваивается переменной Ans.

```
Ans = MsgBox("Продолжить?", vbYesNo + vbQuestion, "Сообщи")
If Ans = vbNo Then Exit Sub
```

Обратите внимание, что в качестве значения аргумента кнопки используется сумма двух встроенных констант (vbYesNo+vbQuestion). Благодаря константе vbYesNo в окне сообщения отображаются две кнопки: одна с меткой Yes, а вторая — с меткой No. Добавление vbQuestion в состав аргумента также приведет к отображению значка вопроса. Как только будет выполнен первый оператор, переменная Ans получит одно из двух значений, представленных константами vbYes и vbNo. В этом примере процедура завершает свою работу после щелчка на кнопке No.

Для получения дополнительных сведений о функции MsgBox обратитесь к главе 12.

В следующем примере показано, каким образом функция рабочего листа Excel используется в процедуре VBA. Нечасто используемая функция VBA Roman преобразует десятичные числа в римские.

```
Sub ShowRoman()
   Dim DecValue As Long
   Dim RomanValue As String
   DecValue = 1939
   RomanValue = Application.WorksheetFunction.Roman(DecValue)
   MsgBox RomanValue
End Sub
```

При выполнении данной процедуры функция MsgBox отображает строку MCMXXXIX. Любители старых фильмов часто приходят в отчаяние, когда узнают, что Excel не располагает функцией, преобразующей римское число в его десятичный эквивалент.

Важно понимать, что вы не можете использовать функции Excel, для которых в VBA представлены эквивалентные функции. Например, VBA не позволяет получить доступ к функции Excel КОРЕНЬ (SQRT), так как в VBA имеется собственная версия этой функции: Sqr. Таким образом, следующий оператор выдает ошибку:

```
MsgBox Application.WorksheetFunction.Sqrt(123) 'ошибка
```

Перекрестная ссылка

Как описывалось в главе 10, можно воспользоваться VBA для создания пользовательских функций рабочего листа, которые работают подобно встроенным функциям рабочего листа Excel.

Управление объектами и коллекциями

Как программист вы проведете много времени, работая с объектами и коллекциями. Поэтому нужно знать самые эффективные способы написания кода для обработки объектов и коллекций. VBA предлагает две конструкции, которые помогут вам упростить управление объектами и коллекциями:

- конструкция `With - End With`;
- конструкция `For Each - Next`.

Конструкция `With - End With`

Конструкция `With - End With` позволяет выполнять несколько операций над одним объектом. Чтобы понять, как она работает, проанализируйте следующую процедуру, которая изменяет шесть свойств выделенного объекта (подразумевается, что выделен объект `Range`).

```
Sub ChangeFont1()
   Selection.Font.Name = "Cambria"
   Selection.Font.Bold = True
   Selection.Font.Italic = True
   Selection.Font.Size = 12
   Selection.Font.Underline = xlUnderlineStyleSingle
   Selection.Font.ThemeColor = xlThemeColorAccent1
End Sub
```

Эту процедуру можно переписать с помощью конструкции `With - End With`. Процедура, показанная ниже, работает точно так же, как и предыдущая.

```
Sub ChangeFont2()
   With Selection.Font
      .Name = "Cambria"
      .Bold = True
      .Italic = True
      .Size = 12
      .Underline = xlUnderlineStyleSingle
      .ThemeColor = xlThemeColorAccent1
   End With
End Sub
```

Считается, что второй вариант этой процедуры читать сложнее. Однако помните, что целью изменений является увеличение скорости выполнения операций. Первый вариант более прямолинейный и его легче понять, но процедура, использующая для изменения нескольких свойств одного объекта конструкцию `With - End With`, помогает повысить эффективность выполнения кода по сравнению с эквивалентной ей процедурой, которая явно ссылается на объект в каждом операторе.

Примечание

При записи макроса VBA в Excel конструкция `With - End With` применяется при каждой удобной возможности. Чтобы посмотреть удачный пример этой конструкции, проанализируйте последовательность действий при изменении параметров страницы с помощью команды Разметка страницы⇨Параметры страницы⇨Ориентация (Page Layout⇨Page Setup⇨Orientation).

Конструкция For Each - Next

Итак, вы уже знаете, что *коллекция* — это группа однородных объектов. Например, Workbooks — это коллекция всех открытых рабочих книг (объектов) Workbook. Существует ряд других коллекций, с которыми вы можете работать.

Предположим, вы решили выполнить действие над всеми объектами коллекции или вам необходимо оценить все объекты коллекции и совершить действие при выполнении определенных условий. Это идеальная ситуация для применения конструкции For Each - Next.

Синтаксис конструкции For Each - Next приведен ниже.

```
For Each элемент In коллекция
   [инструкции]
   [Exit For]
   [инструкции]
Next [элемент]
```

В процедуре, приведенной далее, применена конструкция For Each - Next по отношению к коллекции Worksheets активной рабочей книги. При выполнении этой процедуры функция MsgBox отображает свойство Name каждого рабочего листа. (Если в активной рабочей книге пять листов, функция MsgBox вызывается пять раз.)

```
Sub CountSheets()
   Dim Item as Worksheet
   For Each Item In ActiveWorkbook.Worksheets
      MsgBox Item.Name
   Next Item
End Sub
```

Примечание

В предыдущем примере Item — это объектная переменная (точнее, объект Worksheet). В названии Item не заключен особый смысл; вместо него можно использовать любое корректное название переменной.

В следующем примере конструкция For Each - Next используется для циклического просмотра всех объектов коллекции Windows, а также для подсчета количества скрытых окон.

```
Sub HiddenWindows()
   Dim Cnt As Integer
   Dim Win As Window
   Cnt = 0
   For Each Win In Windows
      If Not Win.Visible Then
   Cnt = Cnt + 1
   Next Win
   MsgBox Cnt & " скрытые окна."
End Sub
```

Для каждого скрытого окна значение переменной Cnt увеличивается на единицу. По завершении цикла в окне сообщения отображается значение переменной Cnt.

В следующем примере закрываются все окна, за исключением активного. В этой процедуре используется конструкция If-Then для проверки каждой рабочей книги в коллекции Workbooks.

```
Sub CloseInactive()
   Dim Book as Workbook
   For Each Book In Workbooks
      If Book.Name <> ActiveWorkbook.Name Then Book.Close
   Next Book
End Sub
```

Чаще всего конструкция `For Each - Next` применяется для циклического обхода всех ячеек диапазона. В следующем примере конструкция `For Each - Next` вызывается после того, как пользователь выбрал диапазон ячеек. В данном случае объект `Selection` выступает в качестве коллекции, которая состоит из объектов `Range`, поскольку каждая ячейка в выделенной области представляет собой объект `Range`. Процедура проверяет каждую ячейку, а затем использует функцию VBA `UCase` для преобразования ее содержимого в символы верхнего регистра. (При этом числовые ячейки не изменяются.)

```
Sub MakeUpperCase()
   Dim Cell as Range
   For Each Cell In Selection
      Cell.Value = UCase(Cell.Value)
   Next Cell
End Sub
```

В VBA существует способ выхода из цикла `For-Next` до того, как будут проверены все элементы коллекции. Для этого используется оператор `Exit For`. В приведенном ниже примере выбирается первое отрицательное значение в первой строке активного листа.

```
Sub SelectNegative()
   Dim Cell As Range
   For Each Cell In Range("1:1")
      If Cell.Value < 0 Then
         Cell.Select
         Exit For
      End If
   Next Cell
End Sub
```

В рассматриваемом примере используется конструкция `If-Then` для проверки значения каждой ячейки. Если ячейка содержит отрицательную величину, она выбирается, причем завершение цикла происходит после выполнения оператора `Exit For`.

Контроль за выполнением кода

Некоторые процедуры VBA начинают выполняться с первых строк кода. Этот процесс происходит построчно до самого конца процедуры (например, так всегда работают макросы, записанные при выполнении действий). Однако иногда необходимо контролировать последовательность операций, пропуская отдельные операторы, повторно выполняя некоторые команды и проверяя условия для определения следующего действия, выполняемого процедурой.

В предыдущем разделе описана конструкция `For Each-Next`, которая является циклической структурой. В настоящем разделе речь пойдет о дополнительных способах контроля за выполнением процедур, написанных на VBA:

- оператор `GoTo`;
- конструкция `If-Then`;

- конструкция Select Case;
- цикл For-Next;
- цикл Do While;
- цикл Do Until.

Оператор GoTo

Самый простой способ изменить последовательность операций в коде — использовать оператор GoTo. Он перенаправляет ход выполнения программы на новую инструкцию, которая помечена специальным образом (текстовая строка, заканчивающаяся двоеточием, или число, заканчивающееся пробелом, указанные перед инструкцией). Процедуры VBA могут содержать любое количество меток, а оператор GoTo не определяет переход за пределы процедуры.

В приведенной ниже процедуре применена функция VBA InputBox для получения имени пользователя. Если имя пользователя отличается от Ховард, то процедура переходит к метке WrongName, на чем заканчивает свою работу. В противном случае процедура выполняет дополнительные операции. Оператор Exit Sub заканчивает выполнение процедуры.

```
Sub GoToDemo()
   UserName = InputBox("Введите свое имя:")
   If UserName <> "Ховард" Then GoTo WrongName
   MsgBox ("Привет, Ховард...")
   ' -[Здесь вводится дополнительный код] -
   Exit Sub
WrongName:
   MsgBox "Извините, эту процедуру может запускать только Ховард."
End Sub
```

Представленная процедура работает, но оператор GoTo, как правило, используется, если другого способа выполнить действие просто не существует. Единственной ситуацией, когда оператор GoTo в VBA действительно необходим, является перехват ошибок (см. главу 9).

Вряд ли представленный выше пример следует рассматривать в качестве эффективной меры защиты!

Конструкция If-Then

Вероятно, конструкция If-Then чаще остальных используется для группирования инструкций VBA. Эта популярная конструкция наделяет приложения способностью принимать решения. Такая способность является ключевой при создании эффективных программ. Удачное приложение Excel, по сути, сводится к принятию правильного решения и выполнению соответствующих действий.

Стандартный синтаксис конструкции If-Then таков:

```
If условие Then инструкции_истина [Else инструкции_ложь]
```

Конструкция If-Then используется для выполнения одного или более операторов при справедливости заданного условия. Оператор Else необязателен. Он позволяет выполнять одну или более инструкций в случае несправедливости условия.

В описанной ниже процедуре применена структура `If-Then` без оператора `Else`. Пример связан с управлением временными данными. VBA использует систему дат и времени, похожую на задействованную в Excel. Время дня выражается дробным числом, например полдень представлен как `0.5`. Функция VBA `Time` возвращает значение, представляющее время в формате системных часов. В следующем примере сообщение отображается, если текущее время меньше полудня. Если текущее системное время больше или равно `0.5`, то процедура заканчивается, и ничего не происходит.

```
Sub GreetMe1()
   If Time < 0.5 Then MsgBox "Доброе утро "
End Sub
```

Можно переписать этот код, воспользовавшись несколькими операторами, как показано ниже.

```
Sub GreetMe1a()
   If Time < 0.5 Then
      MsgBox "Доброе утро"
   End If
End Sub
```

Обратите внимание, что оператору `If` соответствует оператор `End If`. В этом примере вызывается на выполнение только один оператор, если условие равно `True`. Между операторами `If` и `End If` можно поместить любое количество операторов.

Если нужно отобразить другое приветствие, когда наступает вторая половина дня, добавьте еще один оператор `If-Then`, как показано ниже.

```
Sub GreetMe2()
   If Time < 0.5 Then MsgBox "Доброе утро"
   If Time >= 0.5 Then MsgBox "Добрый день"
End Sub
```

Обратите внимание, что для второй конструкции `If-Then` использован оператор `>=` (больше или равно). Таким образом учитывается случай, когда время на часах точно равно 12.00.

Другой подход — использовать оператор `Else` конструкции `If-Then`.

```
Sub GreetMe3()
   If Time < 0.5 Then MsgBox "Доброе утро" Else _
      MsgBox "Добрый день"
End Sub
```

Обратите внимание, что был использован символ продолжения строки; `If-Then-Else` является фактически одним оператором.

Если нужно выполнять несколько операторов на базе одного условия, воспользуйтесь следующим кодом.

```
Sub GreetMe3a()
   If Time < 0.5 Then
      MsgBox "Доброе утро"
      ' Здесь вводятся другие операторы
   Else
      MsgBox "Добрый день "
      ' Здесь вводятся другие операторы
   End If
End Sub
```

Если вам необходимо расширить процедуру до обработки трех условий (например, утро, день и вечер), то можете использовать либо три оператора `If-Then`, либо вложенную структуру `If-Then-Else`. Первый вариант значительно проще.

```
Sub GreetMe4()
   If Time < 0.5 Then MsgBox "Доброе утро"
   If Time >= 0.5 And Time < 0.75 Then MsgBox "Добрый день"
   If Time >= 0.75 Then MsgBox "Добрый вечер"
End Sub
```

Значение `0.75` представляет время 18:00 — три четверти суток и тот момент, когда день переходит в вечер.

В предыдущих примерах каждая инструкция в процедуре выполняется даже утром. Для эффективности следует включить структуру, заканчивающую процедуру, когда одно из условий выполняется. Если, например, отображается сообщение `Доброе утро`, то процедура заканчивается без проверки других, излишних условий. Конечно, разница в скорости выполнения обеих процедур несущественна, если вы разрабатываете такую крохотную процедуру, как рассматриваемая. Но в более сложных приложениях все же рекомендуется учесть следующий синтаксис.

```
If условие Then
   [инструкции_истина]
[ElseIf условие-n Then
   [альтернативные_инструкции]]
[Else
   [операторы_по_умолчанию]]
End If
```

Этот же синтаксис можно использовать для введения кода процедуры `GreetMe5`.

```
Sub GreetMe5()
   If Time < 0.5 Then
      MsgBox "Доброе утро"
   ElseIf Time >= 0.5 And Time < 0.75 Then
      MsgBox "Добрый день"
   Else
      MsgBox "Добрый вечер"
   End If
End Sub
```

Если в представленном синтаксисе условие выполняется, то выполняются соответствующие операторы, после чего конструкция `If-Then` заканчивается. Другими словами, при этом не оцениваются дополнительные условия. Такой синтаксис наиболее эффективен, однако многим эта программа может показаться сложной для понимания.

Далее представлен еще один способ реализовать рассматриваемый пример. В нем используются вложенные конструкции `If-Then-Else` (без `ElseIf`). Данная процедура также эффективна, и ее легко понять. Обратите внимание, что для каждого оператора `If` существует свой оператор `End If`.

```
Sub GreetMe6()
   If Time < 0.5 Then
      MsgBox "Доброе утро"
   Else
      If Time >= 0.5 And Time < 0.75 Then
         MsgBox "Добрый день"
      Else
         If Time >= 0.75 Then
            MsgBox "Добрый вечер"
```

```
        End If
     End If
  End If
End Sub
```

Ниже продемонстрирован еще один пример, использующий простую форму конструкции `If-Then`. Процедура запрашивает у пользователя значение переменной `Quantity` и отображает скидку на основе полученного значения. Если в окне `InputBox` пользователь щелкнет на кнопке **Отмена**, то переменная `Quantity` приравняется к пустой строке, в результате чего процедура будет завершена. Следует отметить, что эта процедура не выполняет других проверок (например, мы не проверяем в данном случае введенное числовое значение на отрицательность).

```
Sub Discount1()
  Dim Quantity As Variant
  Dim Discount As Double
  Quantity = InputBox("Введите значение: ")
  If Quantity = "" Then Exit Sub
  If Quantity >= 0 Then Discount = 0.1
  If Quantity >= 25 Then Discount = 0.15
  If Quantity >= 50 Then Discount = 0.2
  If Quantity >= 75 Then Discount = 0.25
  MsgBox "Скидка: " & Discount
End Sub
```

Обратите внимание, что каждый оператор `If-Then` в представленной процедуре всегда выполняется, а значение `Discount` может изменяться. Однако в результате всегда отображается нужное значение.

Следующая процедура — это вариант предыдущей, переписанной с использованием альтернативного синтаксиса. Процедура будет окончена после выполнения блока `True`.

```
Sub Discount2()
  Dim Quantity As Variant
  Dim Discount As Double
  Quantity = InputBox("Укажите количество: ")
  If Quantity = "" Then Exit Sub
  If Quantity >= 0 And Quantity < 25 Then
     Discount = 0.1
  ElseIf Quantity < 50 Then
     Discount = 0.15
  ElseIf Quantity < 75 Then
     Discount = 0.2
  Else
     Discount = 0.25
  End If
  MsgBox "Скидка: " & Discount
End Sub
```

Вложенные структуры `If-Then` достаточно громоздкие. Поэтому рекомендуется использовать их только для принятия простых бинарных решений. Если же необходимо выбрать между тремя и более вариантами, то целесообразно обратиться к конструкции `Select Case`, которая рассматривается далее.

Функция VBA IIf

В VBA существует альтернатива конструкции If-Then: функция IIf. Эта функция имеет три аргумента и работает подобно функции рабочего листа Excel ЕСЛИ. Рассмотрим ее синтаксис:

```
IIf(выражение, часть_True, часть_False)
```

- *Выражение* (обязательный аргумент) — выражение, которое нужно оценить
- *Часть_True* (обязательный аргумент) — значение или выражение, возвращаемое функцией, когда выражение имеет тип True
- *Часть_False* (обязательный аргумент) — значение или выражение, возвращаемое функцией, когда выражение имеет тип False

Следующая инструкция демонстрирует применение функции IIf. В окне сообщений отображается Нуль, если ячейка A1 содержит нуль или пуста, Не-нуль, если ячейка A1 содержит другое значение.

```
MsgBox IIf(Range("A1") = 0, "Нуль", "Не-нуль")
```

Важно понимать, что третий аргумент (*часть_False*) всегда оценивается, даже если второй аргумент (*часть_True*) выполняется. Следовательно, следующий оператор выдает ошибку, если значение n равно нулю.

```
MsgBox IIf(n = 0, 0, 1 / n)
```

Конструкция Select Case

Конструкция Select Case применяется при выборе между тремя и более вариантами. Она справедлива также для двух вариантов и является хорошей альтернативой структуре If-Then-Else. Конструкция Select Case имеет следующий синтаксис.

```
Select Case тестируемое_выражение
   [Case список условий-n
      [операторы-n]]
   [Case Else
      [операторы_по_умолчанию]]
End Select
```

Приведем пример конструкции Select Case, который показывает еще один способ запрограммировать процедуру GreetMe, рассмотренную в предыдущем разделе.

```
Sub GreetMe()
   Dim Msg As String
   Select Case Time
      Case Is < 0.5
         Msg = "Доброе утро"
      Case 0.5 To 0.75
         Msg = "Добрый день"
      Case Else
         Msg = "Добрый вечер"
   End Select
   MsgBox Msg
End Sub
```

Ниже приведена версия процедуры Discount, переписанная с использованием конструкции Select Case. Эта процедура предполагает, что Quantity всегда выражается целым числом. Чтобы упростить процедуру, в ней не выполняется проверка введенных данных.

```
Sub Discount3()
   Dim Quantity As Variant
   Dim Discount As Double
   Quantity = InputBox("Введите значение: ")
   Select Case Quantity
      Case ""
         Exit Sub
      Case 0 To 24
         Discount = 0.1
      Case 25 To 49
         Discount = 0.15
      Case 50 To 74
         Discount = 0.2
      Case Is >= 75
         Discount = 0.25
   End Select
   MsgBox "Скидка: " & Discount
End Sub
```

В операторе `Case` можно использовать запятую, разделяющую несколько значений для одного варианта. В следующей процедуре используется функция VBA `WeekDay`, с помощью которой определяется, является ли текущий день субботой либо воскресеньем (функция `Weekday` возвращает значение 1 либо 7). Затем отображается соответствующее сообщение.

```
Sub GreetUser1()
   Select Case Weekday(Now)
      Case 1, 7
         MsgBox "Это выходные"
      Case Else
         MsgBox "Это не выходные"
   End Select
End Sub
```

Следующий пример демонстрирует еще один способ кодирования предыдущей процедуры.

```
Sub GreetUser2()
   Select Case Weekday(Now)
      Case 2, 3, 4, 5, 6
         MsgBox "Это не выходные"
      Case Else
         MsgBox "Это выходные"
   End Select
End Sub
```

Под каждым оператором `Case` может указываться любое количество инструкций, и они выполняются при условии `Case`, имеющем значение `True`. Если вы используете в каждом случае `Case` только одну инструкцию (как в предыдущем примере), то можете поместить ее в той же строке, что и ключевое слово `Case` (но не забудьте о символе разделения операторов в VBA — двоеточии). Этот прием сделает текст программы более компактным.

```
Sub Discount3()
   Dim Quantity As Variant
   Dim Discount As Double
   Quantity = InputBox("Ввведите количество: ")
   Select Case Quantity
      Case "": Exit Sub
```

```
      Case 0 To 24: Discount = 0.1
      Case 25 To 49: Discount = 0.15
      Case 50 To 74: Discount = 0.2
      Case Is >= 75: Discount = 0.25
   End Select
   MsgBox "Скидка: " & Discount
End Sub
```

Совет

Интерпретатор VBA осуществляет выход из конструкции `Select Case`, как только найдено условие `True`. Следовательно, для максимальной эффективности, в первую очередь, следует выполнить проверку наиболее вероятного случая.

Структуры `Select Case` можно вкладывать друг в друга. Например, следующая процедура использует процедуру VBA `TypeName` для определения того, что выделено в настоящий момент (диапазон ячеек, ничего или что-либо еще). Если выделен диапазон, процедура вызывает на выполнение вложенный оператор `Select Case`, а также проверяет количество ячеек в диапазоне. Если выделена одна ячейка, отображается сообщение `Выделена одна ячейка`. В противном случае отображается сообщение, в котором указывается количество выделенных строк.

```
Sub SelectionType()
   Select Case TypeName(Selection)
      Case "Диапазон"
         Select Case Selection.Count
         Case 1
            MsgBox "Выделена одна ячейка"
         Case Else
            MsgBox Selection.Rows.Count & " строк"
         End Select
      Case "Ничего"
         MsgBox "Ничего не выделено"
      Case Else
         MsgBox "Выделен объект, отличный от диапазона"
   End Select
End Sub
```

Можно создавать конструкции `Select Case` любой степени вложенности, но убедитесь, что каждому оператору `Select Case` соответствует свой оператор `End Select`.

Данная процедура демонстрирует, насколько важно использовать отступы в коде, чтобы выделить его структуру. Например, рассмотрим ту же процедуру, но без отступов.

```
Sub SelectionType()
Select Case TypeName(Selection)
Case "Диапазон"
Select Case Selection.Count
Case 1
MsgBox "Выделена одна ячейка"
Case Else
MsgBox Selection.Rows.Count & " строк"
End Select
Case "Ничего"
MsgBox "Ничего не выделено"
Case Else
MsgBox "Выделен объект, отличный от диапазона"
```

```
End Select
End Sub
```

В таком представлении мало что понятно даже опытному программисту!

Циклическая обработка инструкций

Цикл — это процесс повторения набора инструкций. Возможно, вы заранее знаете, сколько раз должен повториться цикл, или это значение определяется переменными в программе.

Следующий код, в котором в диапазон вводятся последовательные числа, является примером того, что называют плохим циклом. Процедура использует две переменные для хранения начального значения (`StartVal`) и общего количества ячеек, которые необходимо заполнить (`NumToFill`). В этом цикле для управления порядком выполнения операций используется оператор `GoTo`. Если переменная `Cnt`, отвечающая за то, сколько ячеек заполнено, меньше числа, заданного пользователем, то выполняется переход назад, к метке `DoAnother`.

```
Sub BadLoop()
   Dim StartVal As Integer
   Dim NumToFill As Integer
   Dim Cnt As Integer
   StartVal = 1
   NumToFill = 100
   ActiveCell.Value = StartVal
   Cnt = 1
DoAnother:
ActiveCell.Offset(Cnt, 0).Value = StartVal + Cnt
   Cnt = Cnt + 1
   If Cnt < NumToFill Then GoTo DoAnother Else Exit Sub
End Sub
```

Описанная процедура выполняется правильно. Почему же она является примером плохого цикла? Настоящие программисты не используют оператор `GoTo`, если можно обойтись без него. Применение операторов `GoTo` в цикле противоречит концепции структурного программирования (см. врезку "Несколько слов о структурном программировании"). Этот оператор значительно усложняет восприятие кода, поскольку в данном случае практически невозможно структурировать цикл с помощью отступов. Кроме того, такой тип неструктурированного цикла делает процедуру подверженной частым ошибкам. Также использование большого количества меток приводит к получению программы с плохой структурой (или без структуры вообще) и бессистемным порядком следования операций.

Поскольку в VBA встроено несколько структурированных команд циклов, в процессе принятия решений практически никогда не требуется прибегать к операторам `GoTo`.

Несколько слов о структурном программировании

Пообщайтесь с профессиональными программистами, и рано или поздно вы услышите термин "структурное программирование". Кроме того, вы обнаружите, что структурные программы лучше иных типов программ.

Что же представляет собой структурное программирование и как его применить к VBA?

Основное условие заключается в том, что процедура (или сегмент программы) должна иметь только одну точку входа и одну точку выхода. Другими словами, тело кода является независимым элементом, а контроль за выполнением программы не должен

выходить за рамки этого элемента. Поэтому в структурной программе не допускается применение оператора GoTo. В подобной программе выполнение кода происходит в определенном порядке, который легко отслеживается (в отличие от кода, в котором программа "перескакивает" с места на место).

Структурную программу проще воспринимать и анализировать, чем неструктурную. И что более важно, ее легче изменять.

VBA — структурированный язык. Он предлагает стандартные структурные конструкции, такие как If-Then-Else и Select Case, а также циклы For-Next, Do Until и Do While. Более того, VBA полностью поддерживает модульный подход к созданию программ.

Если вы начинаете программировать, привыкайте сразу же создавать структурированные программы.

Циклы For-Next

Простейший пример хорошего цикла — For-Next (он уже применялся в нескольких предыдущих примерах). Этот оператор имеет следующий синтаксис.

```
For счетчик = начало To конец [Step шаг]
   [инструкции]
   [Exit For]
   [инструкции]
Next [счетчик]
```

Ниже приведен пример цикла For-Next, в котором не используется значение шаг переменной Step и необязательный оператор Exit For. Эта процедура выполняет оператор Sum = Sum + Sqr(Count) 100 раз и отображает результат — сумму квадратных корней первых 100 целых чисел.

```
Sub SumSquareRoots()
   Dim Sum As Double
   Dim Count As Integer
   Sum = 0
   For Count = 1 To 100
      Sum = Sum + Sqr(Count)
   Next Count
   MsgBox Sum
End Sub
```

В данном примере Count (переменная-счетчик цикла) имеет начальное значение 1 и увеличивается на 1 при каждом повторении цикла. Переменная Sum суммирует квадратные корни каждого значения Count.

Предупреждение

Используя циклы For-Next, важно понимать, что в качестве счетчика цикла используется обычная переменная, и ничего больше. В результате значение счетчика цикла можно изменять в блоке программы, который выполняется между операторами For и Next. Однако это очень неудачный прием, поскольку он может привести к непредсказуемым результатам. Необходимо принять специальные меры предосторожности, чтобы удостовериться в том, что программа не изменяет счетчик цикла.

Можно также указать значение шаг для переменной Step, чтобы пропустить отдельные итерации цикла. Ниже рассмотрена исходная процедура, переписанная так, что суммируются квадратные корни всех нечетных чисел в диапазоне от 1 до 100.

```
Sub SumOddSquareRoots()
   Dim Sum As Double
   Dim Count As Integer
   Sum = 0
   For Count = 1 To 100 Step 2
      Sum = Sum + Sqr(Count)
   Next Count
   MsgBox Sum
End Sub
```

В этой процедуре исходное значение `Count` равно 1, затем счетчик принимает значения 3, 5, 7 и т.д. Последнее значение `Count`, используемое в цикле, равно 99. Когда цикл заканчивается, значение `Count` равно 101.

Значение переменной `Step` в цикле `For-Next` может быть также отрицательным. Приведенная ниже процедура удаляет строки 2, 4, 6, 8 и 10 в активном листе.

```
Sub DeleteRows()
   Dim RowNum As Long
   For RowNum = 10 To 2 Step -2
      Rows(RowNum).Delete
   Next RowNum
End Sub
```

На первый взгляд, вызывает удивление использование отрицательного значения переменной `Step` в процедуре `DeleteRows`. Если же использовать положительные значения переменной `Step`, как показано в следующей процедуре, будут удалены не те строки. Это связано с тем, что строки, находящиеся под удаленными строками, получают новые номера. Например, если удалена 2-я строка, 3-я строка получает номер 2. Эту проблему можно устранить путем использования отрицательных значений переменной `Step`.

```
Sub DeleteRows2()
   Dim RowNum As Long
   For RowNum = 2 To 10 Step 2
      Rows(RowNum).Delete
   Next RowNum
End Sub
```

Следующая процедура выполняет ту же задачу, что и пример цикла `BadLoop`, который приводился в начале раздела "Циклическая обработка инструкций". В данном случае был устранен оператор `GoTo`, благодаря чему "плохой цикл" был превращен в "хороший цикл", в котором используется структура `For-Next`.

```
Sub GoodLoop()
   Dim StartVal As Integer
   Dim NumToFill As Integer
   Dim Cnt As Integer
   StartVal = 1
   NumToFill = 100
   For Cnt = 0 To NumToFill - 1
      ActiveCell.Offset(Cnt, 0).Value = StartVal + Cnt
   Next Cnt
End Sub
```

Циклы `For-Next` могут также содержать один или более операторов `Exit For`. Когда программа встречает этот оператор, то сразу же выходит из цикла, как показано в следующем примере. Эта процедура определяет, какая ячейка столбца A на активном рабочем листе имеет наибольшее значение.

```
Sub ExitForDemo()
   Dim MaxVal As Double
   Dim Row As Long
   MaxVal = Application.WorksheetFunction.Max(Range("A:A"))
   For Row = 1 To 1048576
      If Cells(Row, 1).Value = MaxVal Then
         Exit For
      End If
   Next Row
   MsgBox "Максимальное значение в строке " & Row
   Cells(Row, 1).Activate
End Sub
```

Максимальное значение в столбце вычисляется с помощью функции Excel MAX. Затем это значение присваивается переменной MaxVal. Цикл For-Next проверяет каждую ячейку в столбце. Если определенная ячейка равна MaxVal, оператор Exit For заканчивает процедуру. Однако перед выходом из цикла процедура сообщает пользователю о расположении искомой ячейки и активизирует ее.

Примечание

Пример ExitForDemo представлен с целью продемонстрировать выход из цикла For-Next. Однако это не самый эффективный способ найти максимальное значение в диапазоне. Поставленную задачу может выполнить единственный оператор.

```
Range("A:A").Find(Application.WorksheetFunction.Max _
   (Range("A:A"))).Activate
```

В предыдущих примерах использовались достаточно простые циклы. Но вы можете помещать в цикл любое количество операторов и даже вкладывать циклы For-Next в другие циклы For-Next. Ниже приведен пример, в котором применены вложенные циклы For-Next для инициализации массива 10×10×10 значением –1. По завершении выполнения процедуры все 1000 элементов массива MyArray будут содержать значение –1.

```
Sub NestedLoops()
   Dim MyArray(1 to 10, 1 to 10, 1 to 10)
   Dim i As Integer, j As Integer, k As Integer
   For i = 1 To 10
      For j = 1 To 10
         For k = 1 To 10
            MyArray(i, j, k) = -1
         Next k
      Next j
   Next i
End Sub
```

Циклы Do While

Оператор Do While — еще один тип циклической структуры, представленной в VBA. В отличие от цикла For-Next, цикл Do While выполняется до тех пор, пока удовлетворяется заданное условие. Цикл Do While может иметь один из двух представленных ниже синтаксисов.

```
Do [While условие]
   [инструкции]
   [Exit Do]
```

```
   [инструкции]
Loop
```

или

```
Do
   [инструкции]
   [Exit Do]
   [инструкции]
Loop [While условие]
```

Как видите, VBA позволяет проверять условие `While` в начале или в конце цикла. Разница между этими двумя синтаксисами связана с моментом, когда оценивается условие. В первом синтаксисе содержимое цикла может вообще не выполняться. Во втором содержимое цикла всегда выполняется (как минимум один раз).

В следующих примерах в активный рабочий лист вставляется набор дат. Эти даты соответствуют дням текущего месяца, а их ввод в столбец осуществляется, начиная с активной ячейки.

В этих примерах используются некоторые функции VBA, обрабатывающие даты:

- `Date` возвращает текущую дату;
- `Month` возвращает номер месяца для даты, являющейся аргументом;
- `DateSerial` конструирует дату на основе значений дня, месяца и года, указанных в качестве аргументов.

В первом примере демонстрируется цикл `Do While`, в котором реализована проверка условия в начале цикла: процедура `EnterDates1` вводит даты текущего месяца в столбец рабочего листа, начиная с активной ячейки.

```
Sub EnterDates1()
' цикл Do While, условие проверяется в начале
Dim TheDate As Date
TheDate = DateSerial(Year(Date), Month(Date), 1)
Do While Month(TheDate) = Month(Date)
   ActiveCell = TheDate
      TheDate = TheDate + 1
      ActiveCell.Offset(1, 0).Activate
   Loop
End Sub
```

В этой процедуре используется переменная `TheDate`, которая хранит даты, записанные в рабочем листе. Для инициализации переменной используется первый день текущего месяца. В процессе выполнения цикла значение переменной `TheDate` было введено в активную ячейку, затем это значение было увеличено на единицу, после чего активизируется следующая ячейка. Цикл выполняется до тех пор, пока значение месяца, присвоенное переменной `TheDate`, совпадет со значением месяца текущей даты.

Результат выполнения следующей процедуры будет таким же, что и результат выполнения процедуры `EnterDates1`, но в данном случае используется вторая форма синтаксиса цикла `Do While`, предусматривающая проверку условия в конце цикла.

```
Sub EnterDates2()
' цикл Do While, условие проверяется в конце
   Dim TheDate As Date
   TheDate = DateSerial(Year(Date), Month(Date), 1)
   Do
      ActiveCell = TheDate
```

```
      TheDate = TheDate + 1
      ActiveCell.Offset(1, 0).Activate
   Loop While Month(TheDate) = Month(Date)
End Sub
```

Ниже представлен пример цикла `Do While`. Эта процедура открывает текстовый файл, считывает каждую строку и преобразует текст в символы верхнего регистра. Затем текст сохраняется на активном рабочем листе, начиная с ячейки A1 в направлении вниз до конца столбца. Данная процедура использует функцию VBA `EOF`, которая возвращает значение `True` по достижении конца файла. Последний оператор закрывает текстовый файл.

```
Sub DoWhileDemo1()
   Dim LineCt As Long
   Dim LineOfText As String
   Open "c:\data\textfile.txt" For Input As #1
   LineCt = 0
   Do While Not EOF(1)
      Line Input #1, LineOfText
      Range("A1").Offset(LineCt, 0) = UCase(LineOfText)
      LineCt = LineCt + 1
   Loop
   Close #1
End Sub
```

Перекрестная ссылка

Дополнительные сведения о чтении и записи текстовых файлов с помощью VBA можно найти в главе 27.

Циклы `Do While` также могут включать один или более операторов `Exit Do`. По достижении оператора `Exit Do` цикл завершается, а управление передается оператору, следующему за оператором `Loop`.

Циклы `Do Until`

Структура цикла `Do Until` имеет много общего с конструкцией `Do While`. Разница заключается лишь в том, как проверяется условие цикла. В варианте `Do While` цикл выполняется до тех пор, пока выполняется условие. В цикле `Do Until` цикл выполняется, пока условие не станет выполняться.

Структура `Do Until` может быть представлена двумя вариантами синтаксиса.

```
Do [Until условие]
   [инструкции]
   [Exit Do]
   [инструкции]
Loop
```

или

```
Do
   [инструкции]
   [Exit Do]
   [инструкции]
Loop [Until условие]
```

Ниже показаны два примера кода, которые выполняют те же действия, что и рассмотренные ранее примеры циклов `Do While`. Отличие между этими процедурами заключается в месте проверки условия (в начале либо в конце цикла).

```
Sub EnterDates3()
' Цикл Do Until, проверка условия в начале
  Dim TheDate As Date
  TheDate = DateSerial(Year(Date), Month(Date), 1)
  Do Until Month(TheDate) <> Month(Date)
     ActiveCell = TheDate
     TheDate = TheDate + 1
     ActiveCell.Offset(1, 0).Activate
  Loop
End Sub
Sub EnterDates4()
' Цикл Do Until, проверка условия в конце
  Dim TheDate As Date
  TheDate = DateSerial(Year(Date), Month(Date), 1)
  Do
     ActiveCell = TheDate
     TheDate = TheDate + 1
     ActiveCell.Offset(1, 0).Activate
  Loop Until Month(TheDate) <> Month(Date)
End Sub
```

Следующий пример сначала использовал цикл `Do While`, но позднее был переписан с учетом применения цикла `Until`. В результате изменилась единственная строка, в которой находится оператор `Do`. Получился более "прозрачный" код, не требующий отрицательных значений переменной `Step`, как было в примере с циклом `Do While`.

```
Sub DoUntilDemo1()
  Dim LineCt As Long
  Dim LineOfText As String
  Open "c:\data\textfile.txt" For Input As #1
  LineCt = 0
  Do Until EOF(1)
     Line Input #1, LineOfText
     Range("A1").Offset(LineCt, 0) = UCase(LineOfText)
     LineCt = LineCt + 1
  Loop
  Close #1
End Sub
```

Примечание

В VBA используется еще один вид цикла, `While Wend`. Эта циклическая структура была включена в целях обеспечения совместимости и вряд ли вы встретите ее на практике. Ниже приведен код процедуры ввода данных, в котором используется цикл `While Wend`.

```
Sub EnterDates5()
   Dim TheDate As Date
   TheDate = DateSerial(Year(Date), Month(Date), 1)
   While Month(TheDate) = Month(Date)
      ActiveCell = TheDate
      TheDate = TheDate + 1
      ActiveCell.Offset(1, 0).Activate
   Wend
End Sub
```

Глава 9

Работа с процедурами VBA

В этой главе...

- О процедурах
- Выполнение процедуры
- Передача аргументов процедурам
- Обработка ошибок
- Реальный пример

Процедура содержит группу операторов VBA, которые решают поставленную задачу. Большая часть кода VBA содержится в процедурах. Данная глава посвящена процедурам (Sub), которые выполняют действия, но не возвращают дискретных значений.

О процедурах

Процедура — это последовательность операторов VBA, расположенная в модуле VBA, доступ к которому можно получить с помощью VBE. Модуль может включать любое количество процедур.

Существует несколько способов вызвать, или выполнить, процедуры. Процедура выполняется от начала до конца (этот процесс также можно прервать).

Совет

Процедура может иметь любую длину, но многие пользователи не любят слишком длинных процедур, которые выполняют множество операций. Возможно, проще написать несколько коротких процедур, каждая из которых выполняет свою задачу, а затем создать главную процедуру, которая вызывает меньшие процедуры. Подобный подход упростит дальнейшее управление программой.

Некоторые процедуры получают аргументы. *Аргумент* — это информация, используемая процедурой в процессе выполнения. Аргументы процедуры во многом подобны аргументам, используемым функциями Excel. Инструкции в процедуре обычно выполняют логические операции над аргументами, а результаты процедуры обычно основаны на представляемых ей значениях аргументов.

Перекрестная ссылка

VBA также поддерживает процедуры-функции (Function), которые будут рассмотрены в главе 10. В главе 11 вы найдете ряд дополнительных примеров процедур, как `Sub`, так и `Function`, которые можно использовать в собственных целях.

Объявление процедуры `Sub`

При объявлении процедуры с использованием ключевого слова `Sub` применяется следующий синтаксис.

```
[Private | Public] [Static] Sub имя([список_аргументов])
    [инструкции]
    [Exit Sub]
    [инструкции]
End Sub
```

- `Private` (необязательное ключевое слово). Указывает на то, что процедура доступна только для других процедур в том же модуле.
- `Public` (необязательное ключевое слово). Указывает на то, что процедура доступна для всех остальных процедур во всех модулях рабочей книги. При использовании в модуле, содержащем оператор `Option Private Module`, процедура будет недоступна за пределами проекта.
- `Static` (необязательное ключевое слово). Указывает на то, что переменные процедуры сохраняются после окончания процедуры.
- `Sub` (обязательное ключевое слово). Обозначает начало процедуры.
- *Имя*. Любое корректное название процедуры.
- *Список_аргументов*. Представляет заключенный в скобки список переменных, содержащих аргументы, которые передаются в процедуру. Для разделения аргументов используется запятая. Если процедура не использует аргументы, то необходимо включить в объявление процедуры пустые скобки.
- *Инструкции* (необязательные). Корректные инструкции VBA.
- `Exit Sub` (необязательный оператор). Вызывает немедленный выход из процедуры до ее формального завершения.
- `End Sub` (обязательный оператор). Указывает на завершение процедуры.

Название процедуры

Каждой процедуре нужно присвоить имя. Правила именования процедур в основном те же, что и именования переменных. В идеальном случае название процедуры должно описывать, что выполняют содержащиеся в ней операторы. Лучше всего использовать имена, включающие глагол и существительное (например, `ProcessDate`, `PrintReport`,

Sort_Array либо CheckFilename). Избегайте бессмысленных названий типа DoIt, Update и Fix.

Некоторые программисты используют названия, которые напоминают предложения, описывающие процедуру (например, WriteReportToTextFile — записать отчет в текстовый файл, Get_Print_Options_and_Print_Report — получить параметры печати и напечатать отчет).

Примечание

За некоторыми исключениями, все инструкции VBA, находящиеся в модуле, должны содержаться в процедурах. Исключения касаются объявления переменных уровня модуля, определения пользовательских типов данных, а также некоторых других инструкций, которые определяют параметры на уровне модуля (например, Option Explicit).

Область действия процедуры

В предыдущей главе отмечалось, что область действия переменной определяется модулями и процедурами, в которых может использоваться переменная. Аналогичным образом область действия процедуры определяет, какие процедуры могут ее вызывать.

Процедуры типа Public

По умолчанию все процедуры имеют область действия Public, т.е. могут быть вызваны другими процедурами в любом модуле рабочей книги. При этом ключевое слово Public применять необязательно, хотя программисты включают его для ясности. Приведенные ниже процедуры имеют область действия Public.

```
Sub First()
'   ...[код процедуры]
End Sub
Public Sub Second()
'   ...[код процедуры]
End Sub
```

Процедуры типа Private

Процедуры типа Private могут быть вызваны другими процедурами в этом же модуле, но не процедурами других модулей.

Примечание

В диалоговом окне Макрос (Macro) отображаются только процедуры уровня Public. Следовательно, если имеются процедуры, которые предназначены для вызова другими процедурами в этом же модуле, необходимо объявить их как Private. Благодаря этому предотвращается возможность их запуска из диалогового окна Макрос (Macro).

В следующем примере объявляется процедура типа Private с именем MySub.

```
Private Sub MySub()
'   ... [код процедуры]
End Sub
```

Совет

Всем процедурам модуля можно назначить область действия `Private`, даже если они объявлены с помощью ключевого слова `Public`. Для этого перед первым оператором процедуры помещается следующая конструкция:

```
Option Private Module
```

Если этот оператор добавлен в модуль, можно изъять ключевое слово `Private` из объявления процедуры.

Функция записи макросов Excel обычно создает процедуры с названием `Макрос1`, `Макрос2` и т.д. Все эти процедуры имеют область действия `Public` и не имеют аргументов.

Выполнение процедуры

Далее представлены некоторые способы выполнения, или вызова, процедуры VBA.

- С помощью команды **Run⇨Run Sub/UserForm** (Выполнить⇨Выполнить процедуру/пользовательскую форму) (в VBE). Альтернатива — нажать <F5> либо воспользоваться кнопкой **Run Sub/UserForm** панели инструментов **Standard** (Стандартная).
- Из диалогового окна **Макрос** (Macro) в Excel.
- С помощью комбинации клавиши <Ctrl> и присвоенной процедуре клавиши (если процедуре присвоена комбинация клавиш).
- Щелкнув на кнопке или форме рабочего листа. Для этого кнопке или форме должна быть присвоена процедура.
- Из другой процедуры. Процедуры `Sub` и `Function` могут вызывать другие процедуры.
- С помощью пользовательского элемента управления, находящегося на ленте. Кроме того, встроенные элементы управления ленты могут быть "перенастроены" для вызова макроса на выполнение.
- Из пользовательского контекстного меню.
- После выполнения определенного события. Такими событиями могут выступать открытие рабочей книги, сохранение рабочей книги, закрытие рабочей книги, изменение ячейки, переход на другой рабочий лист и многие другие.
- Из окна отладки (Immediate) в VBE. Просто введите название процедуры, укажите все необходимые аргументы и нажмите клавишу <Enter>.

Рассмотренные методы запуска процедур подробно обсуждаются в следующих разделах.

Примечание

Во многих случаях процедура не будет работать правильно, если она не выполняется в подходящем контексте. Например, если процедура ориентирована на работу с активным рабочим листом, она выдаст сообщение об ошибке (если в качестве активного выбран лист диаграммы). Хорошая процедура включает код, который выполняет проверку контекста, и вместо сообщения об ошибке учтиво сообщает о завершении своей работы.

Выполнение процедуры с помощью команды Run Sub/UserForm

Команда меню Run⇨Run Sub/UserForm (Выполнить⇨Выполнить процедуру/пользовательскую форму) в VBE используется преимущественно для тестирования процедуры в процессе ее разработки. Пользователь вряд ли станет активизировать редактор Visual Basic, чтобы запустить процедуру. Выберите команду Run⇨Run Sub/UserForm (или нажмите клавишу <F5>) в VBE, чтобы выполнить текущую процедуру (другими словами, процедуру, в которой расположен курсор).

Если курсор во время выбора команды Run⇨Run Sub/UserForm находится не в одной из процедур, то VBE отображает диалоговое окно Макрос (Macro), в котором можно выбрать процедуру для выполнения.

Выполнение процедуры в диалоговом окне Макрос

После выбора команды Excel Разработчик⇨Код⇨Макросы (Developer⇨Code⇨Macros) отображается диалоговое окно Макрос (Macro), показанное на рис. 9.1. (Для получения доступа к этому окну нажмите клавиши <Alt+F8>.) В списке Находится в (Macros In) находится область, в которой нужно выбрать отображаемый макрос (например, можно выбрать отображение макросов, находящихся в активной рабочей книге).

Рис. 9.1. Диалоговое окно Макрос

В диалоговом окне Макрос (Macro) не отображается следующее:

- процедуры Sub, которые объявлены с помощью ключевого слова `Private`;
- процедуры Sub, которые требуют одного или более аргументов;
- процедуры Sub, включенные в состав надстроек.

Совет

Хотя процедуры, находящиеся в надстройках, не отображаются в диалоговом окне Макрос (Macro), их можно вызывать на выполнение, если известно имя процедуры. Просто укажите это имя в поле Имя макроса (Macro Name), которое находится в диалоговом окне Макрос (Macro), и щелкните на кнопке Выполнить (Run).

Выполнение процедуры с помощью комбинации клавиш

Любой процедуре, не имеющей аргументов, можно присвоить комбинацию <Ctrl+*специальная клавиша*>. Например, если вы присвоите процедуре с названием `UpdateCustomerList` комбинацию клавиш <Ctrl+U>, то после нажатия <Ctrl+U> она будет выполняться.

В начале записи макроса вам предоставляется возможность задать комбинацию клавиш, которая будет использоваться для его выполнения. Ее можно присвоить в любое удобное для вас время. Чтобы присвоить процедуре комбинацию клавиш (или изменить уже существующую), выполните следующие действия.

1. Вызовите Excel и выберите команду Разработчик⇨Код⇨Макросы (Developer⇨Code⇨Macros).
2. Выберите требуемую процедуру в списке, отображаемом в диалоговом окне Макрос (Macro).
3. Щелкните на кнопке Параметры (Options) для отображения диалогового окна Параметры макроса (Macro Options) (рис. 9.2).
4. Введите символ в поле Ctrl+.

   Примечание. Символ, вводимый в текстовое поле Ctrl+, чувствителен к изменению регистра символов. Если ввести символ **s** (нижний регистр), будет присвоена комбинация клавиш <Ctrl+S>. Если же ввести символ **S** (верхний регистр), будет назначена комбинация клавиш <Ctrl+Shift+S>.

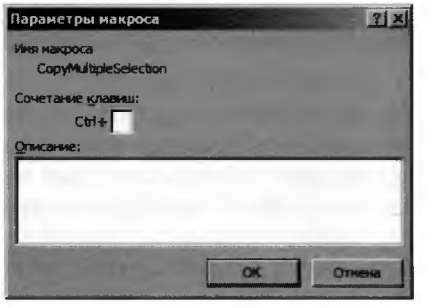


Рис. 9.2. В диалоговом окне Параметры макроса (Macro Options) процедуре можно присвоить комбинацию клавиш и (дополнительно) описание

5. Введите описание (необязательно). Если описание было введено, оно отобразится в нижней части диалогового окна Макрос (Macro) после выбора процедуры в списке.
6. Щелкните на кнопке OK для закрытия диалогового окна Параметры макроса и щелкните на кнопке Закрыть (Close) для закрытия диалогового окна Макрос (Macro).

Предупреждение

Если процедуре назначается одна из предопределенных комбинаций клавиш Excel, ваша команда получает приоритет над этой предопределенной комбинацией. Например, комбинация клавиш <Ctrl+S> соответствует команде сохранения активной рабочей книги в Excel. Но если присвоить эту комбинацию клавиш процедуре, то после нажатия клавиш <Ctrl+S> активная книга больше сохраняться не будет.

Совет

В Excel 2010 не используются следующие клавиши в комбинации <Ctrl+*клавиша*>: E, J, M и Q. В Excel практически не применяются комбинации клавиш <Ctrl+Shift+*клавиша*>, поэтому можно использовать любые из них, за исключением F, L, N, O, P и W.

Выполнение процедуры с помощью ленты

Если вы хотите достичь цели, не прилагая больших усилий, можете написать код XML, выполняющий добавление новой кнопки (либо другого элемента управления) на ленту, а также присвоить макросу этот элемент управления. Обратите внимание, что изменение ленты осуществляется без помощи Excel и VBA.

Новинка

Пользователи Excel 2010 могут изменять ленту с помощью возможности, предоставляемой самой Excel. Теперь можно добавить произвольный элемент управления на ленту, а затем назначить ему макрос.

Перекрестная ссылка

Обратитесь к главе 22 для получения дополнительных сведений о настройке ленты.

Выполнение процедуры из пользовательского контекстного меню

Макрос может быть выполнен путем щелчка мышью на элементе контекстного меню. Это меню отображается после щелчка правой кнопкой мыши на объекте или диапазоне Excel.

Перекрестная ссылка

Дополнительные сведения о контекстных меню можно найти в главе 23.

Выполнение процедуры из другой процедуры

Один из самых популярных способов выполнить процедуру — вызвать ее из другой процедуры. Для этого существует три способа:

- ввести название процедуры и необходимые аргументы этой процедуры (если они есть) через запятую;
- ввести ключевое слово `Call`, после него — название процедуры и ее аргументы (если они есть), заключенные в скобки и разделенные запятыми;
- использовать метод `Run` объекта `Application`. Этот метод можно применять для выполнения других процедур VBA или макросов XLM. Метод `Run` полезен, когда выполняется процедура, название которой присвоено переменной. Тогда переменную можно передать в метод `Run` как аргумент.

В следующем примере показан первый метод. В данном случае процедура `MySub` выполняет некоторые операторы (не показанные в примере), запускает процедуру `UpdateSheet` и затем выполняет остальные операторы.

```
Sub MySub()
'  ...[код]
   UpdateSheet
'  ...[код]
End Sub
```

```
Sub UpdateSheet()
'   ...[код]
End Sub
```

Далее представлен второй метод. Ключевое слово `Call` вызывает процедуру `Update`, имеющую один аргумент; вызывающая процедура передает аргумент в вызываемую процедуру. Аргументы процедур рассматриваются далее.

```
Sub MySub()
   MonthNum = InputBox("Введите номер месяца: ")
   Call UpdateSheet(MonthNum)
'   ... [код]
End Sub

Sub UpdateSheet(MonthSeq)
'   ...[код]
End Sub
```

Совет

Некоторые программисты всегда используют ключевое слово `Call`, чтобы явно указать на вызов другой процедуры, хотя это действие необязательно.

В следующем примере используется метод `Run`, выполняющий процедуру `UpdateSheet` и передающий `MonthNum` в качестве аргумента.

```
Sub MySub()
   MonthNum = InputBox("Введите номер месяца: ")
   Application.Run "UpdateSheet", MonthNum
'   ...[код]
End Sub

Sub UpdateSheet(MonthSeq)
'   ...[код]
End Sub
```

Возможно, главной причиной использования метода `Run` является присвоение названия процедуры переменной. Существует только один способ выполнить процедуру таким образом (см. следующий пример). Процедура `Main` использует функцию VBA `WeekDay` для определения дня недели (целое число от 1 до 7, начиная с воскресенья). Переменной `SubToCall` присваивается строка, содержащая название процедуры. Затем метод `Run` вызывает соответствующую процедуру (`WeekEnd` или `Daily`).

```
Sub Main()
   Dim SubToCall As String
   Select Case WeekDay(Now)
      Case 1, 7: SubToCall = "WeekEnd"
      Case Else: SubToCall = "Daily"
   End Select
      Application.Run SubToCall
End Sub

Sub WeekEnd()
   MsgBox "Сегодня выходной день"
'  Код, который выполняется в выходной
End Sub

Sub Daily()
   MsgBox "Сегодня будний день"
```

```
' Код, который выполняется в будний день
End Sub
```

Вызов процедуры из другого модуля

Если VBA не может найти вызываемую процедуру в текущем модуле, он ищет процедуры `Public` в других модулях этого же проекта.

При вызове процедуры `Private` из другой процедуры обе они должны находиться в одном модуле.

Следует отметить, что в одном модуле не может быть двух процедур с одинаковыми названиями, не допускается также использование процедуры с идентичными именами в разных модулях. Можно указать VBA выполнить неоднозначно названную процедуру, т.е. другую процедуру с тем же названием, но в другом модуле. Для этого введите перед названием процедуры название модуля и точку. Предположим, вы создали процедуры с названием `MySub` в двух модулях: `Module1` и `Module2`. Если требуется, чтобы процедура `MySub` в `Module2` вызывала `MySub` из `Module1`, то используйте один из следующих операторов.

```
Module1.MySub
Call Module1.MySub
```

Если вы не укажете разницу между процедурами с одинаковыми названиями, будет отображено сообщение об ошибке: `Ambiguous Name Detected` (Обнаружено неоднозначное имя).

Вызов процедуры из другой рабочей книги

В некоторых случаях необходимо, чтобы процедура выполняла иную процедуру, определенную в другой рабочей книге. Для этого существует два варианта: установить ссылку на другую рабочую книгу или использовать метод `Run` и явно указать название рабочей книги.

Чтобы добавить ссылку на другую рабочую книгу, выберите в VBE команду **Tools⇨References** (Сервис⇨Ссылки). Будет отображено диалоговое окно **References** (Ссылки), как показано на рис. 9.3, в котором перечислены все существующие ссылки во всех открытых рабочих книгах. Выставьте флажок, соответствующий той рабочей книге, на которую вы планируете добавить ссылку, и щелкните на кнопке **OK**. После установки ссылки процедуры в этой рабочей книге можно вызывать так, будто они находятся в текущей рабочей книге, к которой принадлежит вызывающая процедура.

Рабочая книга, на которую ссылаются, не обязательно должна быть открыта — она рассматривается как отдельная библиотека объектов. Используйте кнопку **Browse** (Просмотр) в диалоговом окне **References**, чтобы установить ссылку на закрытую рабочую книгу.

Примечание

Названия книг, которые указаны в списке ссылок соответствуют названиям проектов VBE. По умолчанию каждый проект обычно называется `VBAProject`. Следовательно, в списке может содержаться несколько одинаково названных элементов. Чтобы отличить определенный проект от других элементов в списке, измените его название в окне **Properties** (Свойства) редактора VBE. Щелкните на имени проекта в окне **Project** (Проект) и выберите команду **Tools⇨*xxx* Properties** (Сервис⇨*xxx* Свойства), где *xxx* — это имя текущего проекта. Находясь в диалоговом окне **Project Properties** (Свойства проекта), щелкните на вкладке **General** (Общие) и измените имя, отображаемое в поле **Project Name** (Имя проекта).

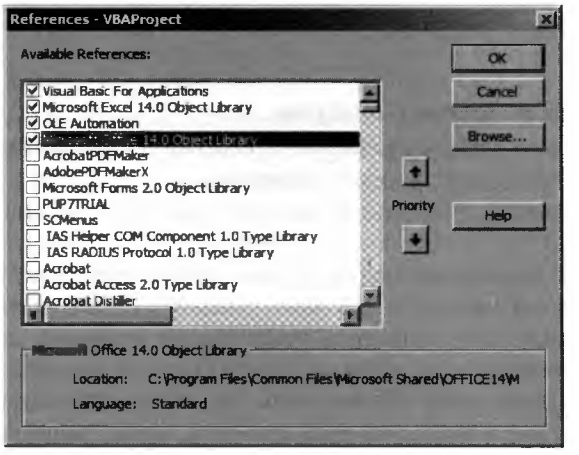


Рис. 9.3. В диалоговом окне References можно установить ссылку на другую рабочую книгу

Список ссылок, который отображается в диалоговом окне References, включает также библиотеки объектов и элементы управления ActiveX, зарегистрированные в вашей системе. Рабочие книги Excel 2010 всегда включают ссылки на следующие библиотеки объектов:

- Visual Basic for Applications;
- Microsoft Excel 14.0 Object Library;
- OLE Automation;
- Microsoft Office 14.0 Object Library;
- Microsoft Forms 2.0 Object Library (необязательно; включается в случае, если в проекте присутствует пользовательская форма (UserForm)).

Примечание

Любые дополнительные ссылки, которые вы добавите, включаются в структуру проекта в окне Project Explorer редактора VBE. Эти ссылки перечислены в узле References.

Если была установлена ссылка на рабочую книгу, которая содержит процедуру `YourSub`, то можете использовать один из следующих операторов для вызова `YourSub`.

```
YourSub
Call YourSub
```

Чтобы точно идентифицировать процедуру в другой рабочей книге, задайте имя проекта, название модуля и имя процедуры, придерживаясь такого синтаксиса:

```
MyProject.MyModule.MySub
```

Можно также использовать ключевое слово `Call`.

```
Call MyProject.MyModule.MySub
```

Причины вызова других процедур

У начинающих программистов нередко вызывает удивление использование метода, подразумевающего вызов процедур из других процедур. Вы можете спросить:

"Не проще ли поместить вызываемую процедуру в вызывающую, упростив тем самым себе жизнь?"

Это делает программу более понятной. Чем проще программа, тем легче ее поддерживать и изменять. Меньшие по размеру процедуры проще понять и отладить. Рассмотрим процедуру, которая лишь вызывает другие процедуры и не выполняет никаких других действий. Разобраться в ней очень легко.

```
Sub Main()
   Call GetUserOptions
   Call ProcessData
   Call CleanUp
   Call CloseItDown
End Sub
```

Вызов других процедур также позволяет избежать повторного ввода кода. Предположим, что операция выполняется в различных местах процедуры. Вместо того чтобы вводить этот код десять раз, можно написать процедуру, выполняющую необходимую операцию, а затем просто ее вызвать десять раз.

Кроме того, вы будете часто обращаться к процедурам универсального назначения. Сохранив их в отдельном модуле, вы сможете импортировать модуль в текущий проект, и при необходимости вызывать эти процедуры. Это значительно проще, чем копировать и вставлять код в новые процедуры.

Зачастую создание нескольких небольших процедур вместо одной большой считается хорошей практикой программирования. Модульный подход не только повышает эффективность работы, но и облегчает жизнь тем людям, которые будут впоследствии работать с вашей программой.

Существует еще один способ вызвать процедуру в другой открытой рабочей книге — использовать метод `Run` объекта `Application`. Этот метод не требует установки ссылки, но содержащая процедуру рабочая книга должна быть открыта. Следующий оператор вызывает процедуру `Consolidate`, которая находится в рабочей книге `budget macros.xlsm`:

```
Application.Run "'budget macros.xlsm'!Consolidate"
```

Выполнение процедуры по щелчку на объекте

Excel содержит ряд объектов, которые можно поместить на рабочем листе или листе диаграммы. Макрос допускается связывать с любым из них. Эти объекты делятся на следующие категории:

- элементы управления ActiveX;
- элементы управления формами;
- включенные объекты (фигуры, рисунки SmartArt, формы WordArt, диаграммы и рисунки).

Примечание

В раскрывающемся списке, вызываемом по команде Разработчик⇨Элементы управления⇨Вставить (Developer⇨Controls⇨Insert), находятся два вида элементов управления, которые можно вставлять на рабочий лист: элементы управления формами и элементы управления ActiveX. Элементы управления ActiveX подобны элементам управления, применяемым в пользовательских формах (UserForm). Элементы управления формами были разработаны для версий Excel 5 и Excel 95, но они могут применяться и в более

поздних версиях (и в некоторых случаях являются предпочтительными). В отличие от элементов управления формами, элементы управления ActiveX не могут применяться для вызова произвольного макроса. Они предназначены для вызова на выполнение макросов специального вида. Например, если на форму добавить элемент управления ActiveX (кнопку) под названием `CommandButton1`, после щелчка на ней вызывается макрос `CommandButton1_Click`, который находится в модуле кода рабочего листа, на котором находится элемент управления. Дополнительные сведения о применении элементов управления на рабочих листах можно найти в главе 13.

Чтобы связать процедуру с объектом `Button` (Кнопка), который находится на панели инструментов **Формы** (Form), выполните следующие действия.

1. Выберите команду **Разработчик⇨Элементы управления⇨Вставить** (Developer⇨Controls⇨Insert) и щелкните на кнопке в группе **Элементы управления формы** (Form Controls).
2. Щелкните на рабочем листе для создания кнопки.

   Для изменения размера кнопки, заданного по умолчанию, перетащите указатель мыши на рабочем листе.

   Excel сразу же отображает диалоговое окно **Назначить макрос** (Assign Macro) (рис. 9.4). Пользователю предлагается воспользоваться макросом, основанным на названии кнопки.
3. Выберите или введите имя макроса, который будет связан с кнопкой, и щелкните на кнопке **OK**.

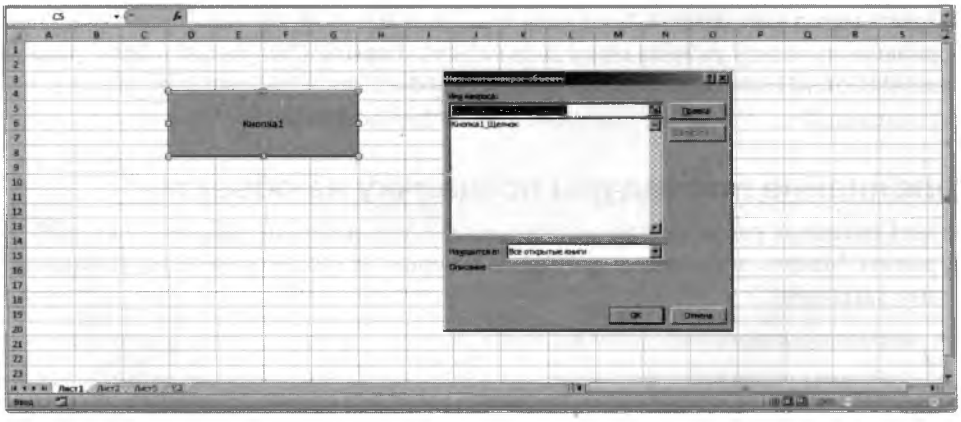


Рис. 9.4. Назначение макроса кнопке

Для изменения назначенного ранее макроса щелкните на кнопке правой кнопкой мыши и выберите команду **Назначить макрос** (Assign Macro).

Для связывания макроса с фигурой, рисунком SmartArt, формой WordArt, диаграммой либо рисунком щелкните правой кнопкой мыши на объекте и в контекстном меню выберите команду **Назначить макрос** (Assign Macro).

Выполнение процедуры по событию

Некоторые процедуры должны выполняться, если в системе (программе) происходит определенное событие. Это может быть открытие рабочей книги, ввод данных в рабочий лист, сохранение книги, щелчок на элементе управления `CommandButton` и многое другое. Процедура, которая выполняется, когда происходит какое-либо событие, называется обработкой события. Процедуры обработки событий характеризуются общими свойствами:

- имеют специальные названия, состоящие из имени объекта, символа подчеркивания и названия события; например, процедура, которая выполняется при открытии рабочей книги, называется `Workbook_Open`;
- хранятся в окне кода для определенного объекта.

Перекрестная ссылка

Процедуры обработки событий описываются в главе 19.

Выполнение процедуры в окне отладки

Процедуру можно также выполнить, введя ее название в окне отладки (Immediate) программы VBE. Если это окно в данный момент не отображено, нажмите <Ctrl+G>. Окно отладки выполняет операторы VBA, которые вы вводите в его области. Чтобы выполнить процедуру, достаточно ввести название процедуры в окне отладки и нажать клавишу <Enter>.

Этот метод часто применяется при разработке процедуры, поскольку в окне отладки можно вводить непосредственные команды, чтобы сразу же увидеть результат их выполнения. Продемонстрируем этот прием.

```
Sub ChangeCase()
   Dim MyString As String
   MyString = "Это тест"
   MyString = UCase(MyString)
   Debug.Print MyString
End Sub
```

На рис. 9.5 показан результат выполнения процедуры `ChangeCase` в окне отладки. Оператор `Debug.Print` немедленно отобразит его.

Передача аргументов процедурам

Аргументы обеспечивают процедуру данными, использующимися в ее инструкциях. Аргумент может передавать следующие данные:

- переменная;
- константа;
- массив;
- объект.

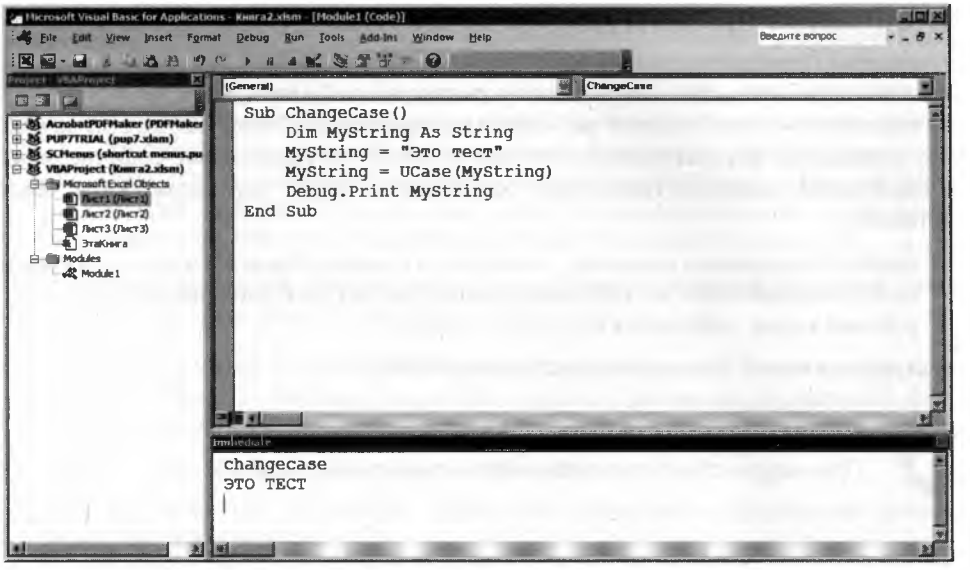


Рис. 9.5. Выполнение процедуры после ввода ее имени в окне отладки

Что касается аргументов, процедуры подобны функциям Excel в следующем:

- процедура может не принимать аргументы;
- процедура может иметь фиксированное количество аргументов;
- процедура может иметь неопределенное количество аргументов;
- не все аргументы процедуры являются обязательными;
- все аргументы могут быть необязательными.

Например, существует несколько функций Excel, не имеющих аргументов (например, СЛЧИС или ТДАТА). Другие функции (такие, как СЧЁТЕСЛИ) принимают два аргумента. Функции еще одной группы (например, СУММ) могут иметь до 255 аргументов. Некоторые функции Excel имеют необязательные аргументы. Например, функция ПЛТ может иметь пять аргументов (три обязательных и два необязательных).

Большинство процедур, с которыми вы сталкивались ранее, объявлялись без аргументов: вводилось ключевое слово Sub, после которого указывались название процедуры и пустые скобки. Пустые скобки означают, что процедура не имеет аргументов.

В примере, приведенном далее, отображаются две процедуры. Процедура Main вызывает процедуру ProcessFile трижды (оператор Call задается в цикле For-Next). Однако перед вызовом ProcessFile создается трехэлементный массив. Внутри цикла каждый элемент массива становится аргументом вызываемой процедуры. Процедура ProcessFile принимает один аргумент (с названием TheFile). Обратите внимание, что аргумент указывается в скобках в операторе Sub. После выполнения ProcessFile программа продолжается с оператора, указанного после оператора Call.

```
Sub Main()
   Dim File(1 To 3) As String
   Dim i as Integer
   File(1) = "dept1.xlsx"
   File(2) = "dept2.xlsx"
```

```
   File(3) = "dept3.xlsx"
   For i = 1 To 3
      Call ProcessFile(File(i))
   Next i
End Sub

Sub ProcessFile(TheFile)
   Workbooks.Open FileName:=TheFile
'   ...[код]
End Sub
```

Конечно, вы всегда можете передавать в процедуру текст (не переменные).

```
Sub Main()
   Call ProcessFile("budget.xlsx")
End Sub
```

Используются два способа передачи аргументов процедуре.

- **По ссылке.** При передаче аргумента по ссылке (метод, применяемый по умолчанию) процедуре передается адрес ячейки памяти, в которой хранится переменная. Поэтому изменение аргумента в процедуре приводит к изменению исходной переменной.
- **По значению.** Передача аргумента по значению фактически означает передачу процедуре копии исходной переменной. Следовательно, изменение аргумента при выполнении процедуры не отражается на исходной переменной.

Следующий пример подтверждает высказанную концепцию. Аргумент процедуры `Process` передается по ссылке (по умолчанию). После того как процедура `Main` присваивает переменной `MyValue` значение `10`, она вызывает процедуру `Process` и передает `MyValue` в качестве аргумента. Процедура `Process` умножает значение своего аргумента (с названием `YourValue`) на 10. По окончании процедуры `Process` возобновляется выполнение процедуры `Main`, а функция `MsgBox` отображает строку `MyValue: 100`.

```
Sub Main()
   Dim MyValue As Integer
   MyValue = 10
   Call Process(MyValue)
   MsgBox MyValue
End Sub

Sub Process(YourValue)
   YourValue = YourValue * 10
End Sub
```

Если требуется, чтобы вызываемая процедура не изменяла переменные, полученные как аргументы, измените список аргументов вызываемой процедуры так, чтобы аргументы передавались по *значению*, а не по *ссылке*. Для этого добавьте перед аргументом ключевое слово `ByVal`. Тогда вызываемая процедура будет управлять копией переданных данных, а не самими данными. В следующей процедуре, например, изменения, которые происходят с `YourValue` в процедуре `Process`, не влияют на значение переменной `MyValue` в процедуре `Main`. В результате функция `MsgBox` отображает 10, а не 100.

```
Sub Process(ByVal YourValue)
   YourValue = YourValue * 10
End Sub
```

В большинстве случаев вполне подходит метод, используемый по умолчанию, — передача аргументов по ссылке. Однако если процедура призвана изменить данные, которые передаются ей в виде аргументов, а первоначальные данные должны остаться неизменными, то необходимо использовать передачу данных по значению.

Аргументы процедуры могут быть самыми разными и передаваться и по значению, и первым способом. Все аргументы, перед которыми указано ключевое слово `ByVal`, передаются по значению, остальные — по ссылке.

Примечание

Если в процедуре применена переменная с пользовательским типом данных, то ее нужно передавать по ссылке. При попытке передать ее по значению происходит ошибка.

Переменные типа `Public` и передача аргументов процедуре

В главе 8 уже отмечалось, что переменная типа `Public` (определенная в верхней части модуля) будет доступной для всех процедур этого модуля. В некоторых случаях предпочтительнее пользоваться переменной типа `Public`, чем передавать переменную в виде аргумента при вызове другой процедуры.

Например, приведенная ниже процедура передает значение `MonthVal` процедуре `ProcessMonth`.

```
Sub MySub()
   Dim MonthVal as Integer
'  ...[код]
   MonthVal = 4
   Call ProcessMonth(MonthVal)
'  ...[код]
End Sub
```

Альтернативный метод выглядит следующим образом.

```
Public MonthVal as Integer

Sub MySub()
'  ...[код]
   MonthVal = 4
   Call ProcessMonth2
'  ...[код]
End Sub
```

Во втором фрагменте кода, из-за того что `MonthVal` является переменной типа `Public`, процедура `ProcessMonth2` имеет к ней доступ, поэтому необходимость в аргументах для этой процедуры отпадает.

Так как в предыдущих примерах не был объявлен тип данных ни для одного аргумента, все аргументы имеют тип данных `Variant`. Но процедура может определять типы данных непосредственно в списке аргументов. Ниже представлен оператор `Sub` для процедуры с двумя аргументами, имеющими различные типы данных. Первый аргумент объявлен как `Integer`, второй — как `String`.

```
Sub Process(Iterations As Integer, TheFile As String)
```

При передаче аргументов в процедуру важно, чтобы данные, которые передаются в качестве аргументов, имели тип данных аргументов. Например, в предыдущем примере

при вызове `Process` и передаче в качестве первого аргумента переменной типа `String` отображается сообщение об ошибке `ByRef argument type mismatch`.

Примечание

Аргументы имеют как обычные процедуры, так и функции. На практике чаще всего аргументы используются функциями. Этот вид процедур рассматривается в главе 10, где вы найдете дополнительные примеры использования аргументов в процедурах, а также описание необязательных аргументов.

Обработка ошибок

Вы, конечно же, догадываетесь, что при выполнении процедуры VBA могут происходить ошибки: синтаксические (которые необходимо исправить перед тем, как выполнять процедуру) и ошибки выполнения (они происходят в процессе выполнения процедуры). В этом разделе рассматривается вторая группа ошибок.

Предупреждение

Для обработки ошибок отключите параметр Break on All Errors (Останавливаться при любой ошибке). Для этого в окне редактора VBE выберите команду Tools⇨Options (Сервис⇨Параметры) и щелкните на вкладке General (Общие) в диалоговом окне Options (Параметры). Если параметр Break on All Errors установлен, VBA игнорирует код обработки ошибок. Поэтому обычно используется параметр Break on Unhandled Errors (Останавливаться при необработанных ошибках).

Как правило, ошибка в процессе выполнения вызывает остановку программы VBA, а пользователь видит диалоговое окно, в котором отображаются номер и описание ошибки. В хорошо написанном приложении вы не столкнетесь с такими служебными сообщениями, поскольку в приложение включены специальные процедуры, перехватывающие ошибки и выполняющие соответствующие действия. В крайнем случае программа обработки ошибок отображает для пользователя более значимое сообщение об ошибке, чем представленное VBA.

Перекрестная ссылка

Все коды ошибок VBA и их описания приведены в приложении В.

Перехват ошибок

Чтобы указать программе, что должно произойти при возникновении ошибки, используется оператор `On Error`. Вы вправе выбрать один из двух вариантов.

- **Проигнорировать ошибку и позволить VBA продолжить выполнение программы.** После этого можно проанализировать объект `Err`, чтобы узнать, какая ошибка произошла, и при необходимости принять меры для ее предотвращения.
- **Перейти к специальному разделу кода для обработки ошибок, чтобы выполнить необходимые действия.** Этот раздел вводится в конце процедуры и обозначается специальной меткой.

Чтобы программа продолжала выполняться после возникновения ошибки, необходимо вставить в код следующий оператор:

```
On Error Resume Next
```

Некоторые ошибки несущественны, и их можно просто игнорировать. Однако сначала следует определить, какая ошибка произошла. При возникновении ошибки можно использовать объект `Err` для определения ее номера. Чтобы отобразить значение `Err.Number` (по умолчанию используется в `Err`), обратитесь к функции `Error`. Например, представленный далее оператор отображает ту же информацию, что и обычное диалоговое окно со сведениями об ошибке Visual Basic (содержит номер ошибки и ее описание).

```
MsgBox "Ошибка " & Err & ": " & Error(Err.Number)
```

На рис. 9.6 показано сообщение об ошибке VBA, а на рис. 9.7 отображена та же ошибка, что и в окне сообщения, но уже в Excel. Конечно, вы можете сделать сообщение об ошибке более значимым для конечных пользователей, используя описательный текст.

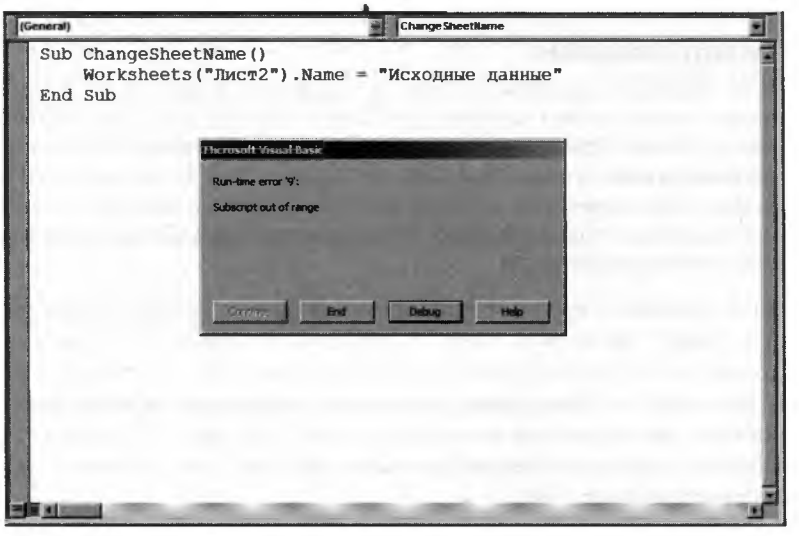


Рис. 9.6. Сообщение об ошибке VBA не всегда понятно пользователям

Примечание

Ссылка на `Err` эквивалентна обращению к свойству `Number` объекта `Err`. Следовательно, два приведенных ниже оператора идентичны.

```
MsgBox Err
MsgBox Err.Number
```

Оператор `On Error` также применяется для определения места в процедуре, к которому должна перейти программа в случае ошибки. Чтобы обозначить это место, используется метка.

```
On Error GoTo ErrorHandler
```

Примеры обработки ошибок

В первом примере демонстрируется ошибка, которую можно спокойно проигнорировать. Метод `SpecialCells` выделяет ячейку, которая соответствует заданному критерию.

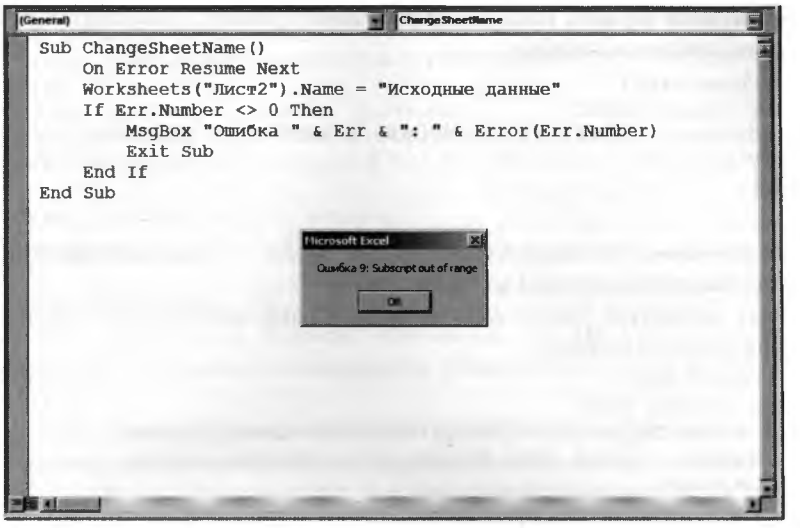


Рис. 9.7. Можно создать более информативное сообщение об ошибке

Примечание

Метод `SpecialCells` выполняет те же действия, что и команда Главная⇨Редактирование⇨Найти и выделить⇨Выделение группы ячеек (Home⇨Editing⇨Find & Select⇨Go To Special). В диалоговом окне Выделение группы ячеек (Go To Special) перед пользователем открываются широкие возможности выбора. Например, можно выделить ячейки, содержащие только числовые константы (не формулы).

В приведенном далее примере метод `SpecialCells` выделяет все ячейки в текущем диапазоне, которые содержат формулу, возвращающую число. Если ни одна ячейка в диапазоне не соответствует критерию, VBA генерирует сообщение об ошибке (рис. 9.8).

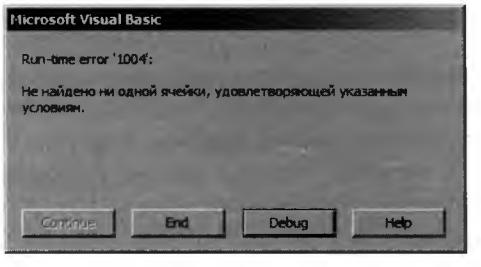


Рис. 9.8. Метод `SpecialCells` генерирует сообщение об ошибке, если нужные ячейки не найдены

```
Sub SelectFormulas()
   Selection.SpecialCells(xlFormulas, xlNumbers).Select

'  ...[код]
End Sub
```

Ниже приводится вариант кода, в котором конструкция `On Error Resume Next` предотвращает появление ошибки.

```
Sub SelectFormulas2()
   On Error Resume Next
   Selection.SpecialCells(xlFormulas, xlNumbers).Select
   On Error GoTo 0
'  ...[код]
End Sub
```

Обратите внимание, что оператор `On Error GoTo 0` восстанавливает нормальную обработку ошибок перед выходом из процедуры.

Следующая процедура использует дополнительный оператор для определения результата: произошла ли ошибка.

```
Sub SelectFormulas2()
   On Error Resume Next
   Selection.SpecialCells(xlFormulas, xlNumbers).Select
   If Err.Number = 1004 Then MsgBox "Не найдены ячейки _
      с формулами."
   On Error GoTo 0
   ' ...[код]
End Sub
```

Если свойство `Number` объекта `Err` не равно `0`, происходит ошибка. С помощью оператора `If` проверяется, не равно ли свойство `Err.Number 1004`, и, если это так, отображается окно сообщения. В рассмотренном примере осуществляется проверка кода на предмет обнаружения ошибки с указанным номером. Если нужно найти все ошибки, используйте следующий оператор:

```
If Err.Number <> 0 Then MsgBox "Произошла ошибка."
```

В следующем примере кода демонстрируется обработка ошибок путем перехода по метке.

```
Sub ErrorDemo()
   On Error GoTo Handler
   Selection.Value = 123
Exit Sub
Handler:
   MsgBox "Невозможно присвоить значение выделенному диапазону."
End Sub
```

В процедуре предпринимается попытка присвоить значение текущему выделенному объекту. Если происходит ошибка (например, не выделен диапазон ячеек или лист защищен), то оператор присваивания выдает ошибку. Оператор `On Error` задает переход к метке `Handler` в случае ошибки. Обратите внимание, что перед меткой используется оператор `Exit Sub`. Программа обработки не выполняется, если ошибок не было. Этот оператор можно и не задавать — в результате сообщение будет отображаться даже в том случае, если ошибка не происходила.

Иногда ошибка помогает получить определенную информацию. В приведенном ниже примере выполняется проверка, открыта ли конкретная рабочая книга. При этом ошибки не обрабатываются.

```
Sub CheckForFile1()
   Dim FileName As String
   Dim FileExists As Boolean
   Dim book As Workbook
   FileName = "BUDGET.XLSX"
   FileExists = False
```

```
' Цикл по всем рабочим книгам
  For Each book In Workbooks
     If UCase(book.Name) = FileName Then FileExists = True
  Next book
' Отображение соответствующего сообщения
  If FileExists Then
     MsgBox FileName & " открыт."
  Else
     MsgBox FileName & " не открыт."
  End If
End Sub
```

В этой процедуре в цикле `For Each-Next` просматриваются все объекты коллекции `Workbooks`. Если книга открыта, переменная `FileExists` получает значение `True`. В результате отображается сообщение, указывающее пользователю, открыта ли рабочая книга.

Предыдущую процедуру можно переписать так, что для определения "открытости" будет использоваться метод обработки ошибок. В следующем примере оператор `On Error Resume Next` указывает VBA игнорировать любые ошибки. В инструкции мы обратимся к рабочей книге, присвоив ей объектную переменную (с помощью ключевого слова `Set`). Если рабочая книга закрыта, происходит ошибка. В структуре `If-Then-Else` проверяется значение свойства `Err` и отображается соответствующее сообщение.

```
Sub CheckForFile()
  Dim FileName As String
  Dim x As Workbook
  FileName = "BUDGET.XLSX"
  On Error Resume Next
  Set x = Workbooks(FileName)
  If Err = 0 Then
     MsgBox FileName & " открыт."
  Else
     MsgBox FileName & " не открыт."
  End If
  On Error GoTo 0
End Sub
```

Перекрестная ссылка

В главе 11 можно найти ряд дополнительных примеров, иллюстрирующих обработку ошибок.

Реальный пример

В этой главе рассмотрены основные принципы создания процедур. Отметим, что многие приведенные примеры были не очень интересными. Далее представлен пример из реальной жизни. Он иллюстрирует многие концепции, рассмотренные в этой и предыдущих двух главах.

В данном разделе описана разработка полезной утилиты, которую можно рассматривать как приложение (согласно определению, предложенному в главе 5). Что более важно, ниже показан процесс анализа задачи и последующего ее решения с помощью VBA. Предупреждение опытным пользователям, читающим эту книгу: примите к сведению, что пример рассмотрен в расчете на начинающих. Поэтому мы не только представим код, но и покажем, какими источниками можно пользоваться при разработке программы.

Компакт-диск

Готовое приложение можно найти на прилагаемом к книге компакт-диске.

Цель

Цель этого упражнения — разработать утилиту, которая изменяет порядок следования листов рабочей книги, сортируя их названия по алфавиту (с помощью одних только функций Excel это сделать невозможно). Если вы часто создаете книги с большим количеством листов, то знаете, что иногда сложно найти интересующий вас лист. Если же их упорядочить по названиям, то любой рабочий лист найти будет значительно проще.

Требования к проекту

С чего начнем? С перечисления требований к приложению. В процессе разработки вы будете обращаться к этому перечню для проверки правильности выполнения действий.

Требования к разрабатываемому приложению приведены далее.

1. Приложение должно сортировать листы (т.е. рабочие листы и листы диаграмм) активной книги по названиям в алфавитном порядке.
2. Приложение должно легко выполняться.
3. Приложение всегда должно быть доступным. Другими словами, пользователь не должен открывать рабочую книгу для использования этой утилиты.
4. Приложение должно правильно выполняться по отношению к любой открытой рабочей книге.
5. В приложении не должны отображаться сообщения об ошибках VBA.

Исходные данные

Часто самой сложной частью проекта является определение того, с чего же начать. В данном случае начнем с перечисления особенностей Excel, которые могут повлиять на соблюдение требований к проекту.

- В Excel отсутствует команда сортировки листов. Следовательно, отпадает вариант записи макроса для упорядочивания листов в алфавитном порядке.
- Лист можно легко переместить, перетащив его за ярлычок.

  Примечание. Включите функцию записи макросов и перетащите лист в другое место, чтобы узнать, какой код создается при таком действии.
- В Excel можно открыть диалоговое окно Переместить или скопировать (Move or Copy), щелкнув правой кнопкой мыши на ярлычке листа с последующим выбором команды контекстного меню Переместить/скопировать (Move or Copy). Будет ли при записи макроса для этой команды генерироваться код, который работает иначе, чем операция перемещения листа вручную?
- Следует знать, сколько листов содержится в активной рабочей книге. Эту информацию можно получить с помощью VBA.
- Узнайте названия листов (вновь воспользовавшись VBA).

- В Excel существует команда, сортирующая данные в ячейках рабочего листа.

  Примечание. Возможно, стоит перенести названия листов в диапазон ячеек и использовать эту функцию. Или, возможно, в VBA есть метод сортировки, которым можно будет воспользоваться в программе.

- Благодаря диалоговому окну **Параметры макроса** (Macro Options) несложно будет назначить для макроса комбинацию клавиш.
- Если макрос сохранен в личной книге макросов, он всегда доступен.
- Вам понадобится тестировать приложение по мере разработки. Естественно, нельзя тестировать приложение в той же рабочей книге, в которой оно разработано.

  Примечание. Создайте фиктивную рабочую книгу, предназначенную специально для тестирования.

- Если разработать программу правильно, то VBA не будет отображать сообщения об ошибках.

  Примечание. Не будем принимать желаемое за действительное...

Подход

На данном этапе мы точно не знаем, что будем делать дальше, однако можно разработать предварительный схематический план, описывающий общие задачи:

1) идентифицировать активную рабочую книгу;
2) получить список названий всех листов в рабочей книге;
3) посчитать листы;
4) отсортировать их (определенным образом);
5) изменить порядок следования листов в соответствии с параметрами сортировки.

Что необходимо знать

В намеченном плане можно заметить несколько недостатков. Необходимо определить, как выполнить следующие операции:

- идентифицировать активную рабочую книгу;
- сосчитать листы активной рабочей книги;
- получить список названий листов;
- отсортировать список;
- изменить порядок следования листов в соответствии с отсортированным списком.

Совет

Если вам недостаточно информации о конкретных методах и свойствах, обратитесь к этой книге или к электронной справочной системе. Вскоре вы найдете то, что вам необходимо. Однако для начала лучше всего включить функцию записи макросов и посмотреть, что записывается в результате выполнения действий, связанных с решением поставленной задачи. Это занятие принесет вам немало пользы.

Некоторые предварительные соображения

Ниже приведен пример, который был получен в результате записи макроса, позволяющей больше узнать о необходимых для решения поставленной задачи средствах VBA. Начнем с рабочей книги, содержащей три рабочих листа. Затем включим функцию записи макросов и укажем, что макрос должен сохраняться в личной книге макросов. В режиме записи перетащим третий рабочий лист на место первого. Результат записи средствами Excel будет следующим.

```
Sub Macro1()
    Sheets("Лист3").Select
    Sheets("Лист3").Move Before:=Sheets(1)
End Sub
```

Найдем в справочной системе слово `Move` (это метод, перемещающий лист в рабочей книге на новое место). Данный метод имеет один аргумент, определяющий будущее положение листа. Таким образом, он имеет непосредственное отношение к нашей задаче.

Вам также необходимо узнать количество листов в активной рабочей книге. Запросим сведения о слове `Count` и определим, что это свойство коллекции. Затем активизируем окно отладки (Immediate) в VBE и введем такой оператор:

```
? ActiveWorkbook.Count
```

Ошибка! Возможно, требуется сосчитать листы в книге.

```
? ActiveWorkbook.Sheets.Count
```

На этот раз успешно. На рис. 9.9 показан результат — еще немного ценной информации.

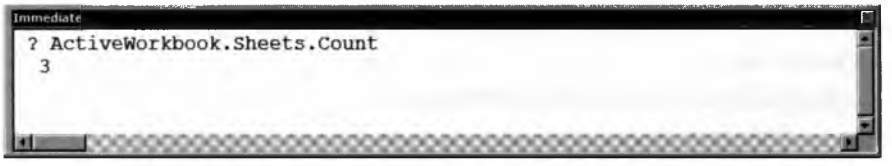


Рис. 9.9. Использование окна отладки в VBE для тестирования оператора

А что с названиями листов? Проведем еще один тест. Введем в окне отладки (Immediate) следующий оператор:

```
? ActiveWorkbook.Sheets(1).Name
```

В результате будет получено название первого листа — `Лист3` (так и есть на самом деле). Таким образом, мы получили положительный результат.

Далее конструкция `For Each-Next` используется для циклического просмотра всех членов коллекции. Просмотрев соответствующий раздел справочной системы, напишем короткую процедуру с использованием этой конструкции.

```
Sub Test()
    For Each Sht In ActiveWorkbook.Sheets
        MsgBox Sht.Name
    Next Sht
End Sub
```

И вновь успешно! Макрос отобразил три окна сообщения, в каждом из которых — новое название листа.

Наконец, пришло время подумать о функциях сортировки. Справочная система укажет, что метод `Sort` относится к объекту `Range`. Поэтому одним из решений задачи будет перенесение названия листов в диапазон ячеек и сортировка этого диапазона. Однако

такая задача оказалась слишком сложной для данного приложения. Возможно, целесообразнее сформировать из названий листов массив строк, а затем отсортировать этот массив с использованием кода VBA.

Подготовка

Теперь мы обладаем достаточным количеством информации, для того чтобы приступить к работе над серьезной программой. Однако прежде следует задать первоначальные настройки. Итак, выполним следующие инструкции.

1. Создайте пустую рабочую книгу с пятью рабочими листами: названия — `Лист1`, `Лист2`, `Лист3`, `Лист4` и `Лист5`.
2. Разместите листы произвольно, чтобы они следовали не по порядку.
3. Сохраните рабочую книгу как `Test.xlsm`.
4. Активизируйте VBE и выберите проект `Personal.xlsb` в окне Project (Проект). Если `Personal.xlsb` не отображается в окне Project, значит, вы никогда не использовали личную книгу макросов. Excel создаст для вас эту книгу, когда вы запишете макрос (любой) и определите, что он должен сохраняться в личной книге макросов.
5. Добавьте новый модуль VBA (используя команду Insert⇨Module (Вставить⇨Модуль)).
6. Создайте пустую процедуру с названием `SortSheets` (рис. 9.10).

   Макрос можно сохранить в любом модуле личной книги макросов. Однако лучше хранить каждый макрос в отдельном модуле. Таким образом, вы сможете легко экспортировать модуль и импортировать его в другой проект.

Рис. 9.10. Пустая процедура в модуле, находящемся в персональной книге макросов

7. Активизируйте Excel. Выберите команду Разработчик⇨Код⇨Макросы (Developer⇨Code⇨Macros) для отображения диалогового окна Макрос (Macro).
8. В диалоговом окне Макрос (Macro) выберите процедуру SortSheets и щелкните на кнопке Параметры (Options), чтобы присвоить данному макросу комбинацию клавиш. Неплохим выбором будет <Ctrl+Shift+S>.

Написание кода

Теперь пришло время приступить к написанию программы. Вначале необходимо поместить названия листов в массив строк. Так как пока неизвестно, сколько листов содержит активная рабочая книга, для объявления массива используем оператор Dim с пустыми скобками. Помните, что всегда можно применить оператор ReDim и изменить размерность массива на требуемое число элементов.

Введем код, вставляющий названия листов в массив SheetNames. Кроме того, в цикл добавим функцию MsgBox, чтобы убедиться, что названия листов на самом деле вводятся в массив.

```
Sub SortSheets()
'  Сортировка листов в активной рабочей книге
   Dim SheetNames() as String
   Dim i as Long
   Dim SheetCount as Long
   SheetCount = ActiveWorkbook.Sheets.Count
   ReDim SheetNames(1 To SheetCount)
   For i = 1 To SheetCount
      SheetNames(i) = ActiveWorkbook.Sheets(i).Name
      MsgBox SheetNames(i)
   Next i
End Sub
```

Чтобы проверить эту программу, активизируем рабочую книгу Test.xlsm и нажмем клавиши <Ctrl+Shift+S>. Появится пять окон сообщений с названиями листов активной рабочей книги.

Рекомендуем вам тестировать код по мере его создания. Когда вы убедитесь, что программа работает правильно, удалите операторы MsgBox (через некоторое время они начнут вас раздражать).

Совет

Вместо того чтобы использовать функцию MsgBox в целях тестирования, можно обратиться к методу Print объекта Debug, который отображает сведения в окне отладки. Для этого замените MsgBox следующим оператором:

```
Debug.Print SheetNames(i)
```

Этот прием не столь навязчив по сравнению с использованием операторов MsgBox. Не забудьте только удалить этот оператор по завершении тестирования.

На данном этапе процедура SortSheets всего лишь создает массив названий листов в соответствии с порядком указания их в активной рабочей книге. Остается два шага: отсортировать значения в массиве SheetNames и изменить порядок следования листов в книге согласно отсортированному массиву.

Создание процедуры сортировки

Переходим к сортировке массива `SheetNames`. Можно вставить программу сортировки в процедуру `SortSheets`, но лучше написать общую процедуру сортировки, которую можно будет использовать для управления другими объектами (сортировка массивов — довольно популярная операция).

Возможно, вас несколько обескуражит идея создания процедуры сортировки. Однако не беспокойтесь: несложно найти стандартные процедуры, которые можно непосредственно или с небольшими изменениями использовать в своей программе. Конечно, наиболее полным источником такой информации является Интернет.

Существует несколько способов сортировки массивов. Мы выбрали пузырьковый метод (хотя это не очень быстрый прием, но его легко запрограммировать). В данном конкретном приложении высокая скорость выполнения операций не так уж важна.

В пузырьковом методе используется вложенный цикл `For-Next`, в котором оценивается каждый элемент массива. Если элемент массива больше, чем следующий, то эти два элемента меняются местами. Такое сравнение повторяется для каждой пары элементов (т.е. n – 1 раз).

Перекрестная ссылка

В главе 11 описаны другие процедуры сортировки, а также приведено сравнение процедур по скорости выполнения.

Ниже приведена написанная мною процедура сортировки (после консультации с разработчиками веб-сайтов, которые подсказали некоторые идеи).

```
Sub BubbleSort(List() As String)
   Dim First As Long, Last As Long
   Dim i As Long, j As Long
   Dim Temp As String
   First = LBound(List)
   Last = UBound(List)
   For i = First To Last - 1
      For j = i + 1 To Last
         If List(i) > List(j) Then
            Temp = List(j)
            List(j) = List(i)
            List(i) = Temp
         End If
      Next j
   Next i
End Sub
```

Эта процедура имеет один аргумент: одномерный массив с названием `List`. Массив, который передается в процедуру, может быть любой длины. Для присвоения нижней и верхней границ массива переменным `First` и `Last` использовались функции `Lbound` и `UBound` соответственно.

Ниже приведена небольшая временная процедура, которая используется для тестирования процедуры `BubbleSort`.

```
Sub SortTester()
   Dim x(1 To 5) As String
   Dim i As Long
   x(1) = "собака"
   x(2) = "кот"
```

```
   x(3) = "слон"
   x(4) = "трубкозуб"
   x(5) = "птица"
   Call BubbleSort(x)
   For i = 1 To 5
      Debug.Print i, x(i)
   Next i
End Sub
```

Процедура `SortTester` создает массив из пяти строк, передает его процедуре `BubbleSort` и отображает отсортированный массив в окне отладки (Immediate). После того как код выполнил свое предназначение, он был удален.

Убедившись в том, что код работает надежно, я изменил процедуру `SortSheets` путем добавления вызова в процедуру `BubbleSort`, передачи массива `SheetNames` в качестве аргумента. Начиная с этого момента, модуль приобретает следующий вид.

```
Sub SortSheets()
   Dim SheetNames() As String
   Dim SheetCount as Long
   Dim i as Long
   SheetCount = ActiveWorkbook.Sheets.Count
   ReDim SheetNames(1 To SheetCount)
   For i = 1 To SheetCount
      SheetNames(i) = ActiveWorkbook.Sheets(i).Name
   Next i
   Call BubbleSort(SheetNames)
End Sub

Sub BubbleSort(List() As String)
'  Сортировка массива List по возрастанию
   Dim First As Long, Last As Long
   Dim i As Long, j As Long
   Dim Temp As String
   First = LBound(List)
   Last = UBound(List)
   For i = First To Last - 1
      For j = i + 1 To Last
         If List(i) > List(j) Then
            Temp = List(j)
            List(j) = List(i)
            List(i) = Temp
         End If
      Next j
   Next i
End Sub
```

По окончании работы процедуры `SortSheets` образуется массив, состоящий из отсортированных названий листов активной рабочей книги. Чтобы проверить это, можно отобразить содержимое массива в окне отладки, добавив в конец процедуры `SortSheets` такой код (если это окно не отображается, нажмите клавиши <Ctrl+G>).

```
For i = 1 To SheetCount
   Debug.Print SheetNames(i)
Next i
```

Пока все идет нормально. Осталось написать программу для изменения порядка следования листов в книге в соответствии с отсортированными элементами массива `SheetNames`.

Программа, записанная раньше, вновь пригодилась. Помните инструкцию, которая была получена при перемещении листа в рабочей книге в первую позицию.

```
Sheets("Лист3").Move Before:=Sheets(1)
```

Далее напишем цикл `For-Next`, который просматривает каждый лист и перемещает его в соответствующее место, указанное в массиве `SheetNames`.

```
For i = 1 To SheetCount
   Sheets(SheetNames(i)).Move Before:=Sheets(i)
Next i
```

Например, в первой итерации цикла счетчик (`i`) равен 1. Первый элемент массива `SheetNames` (в данном примере) — лист `Лист1`. Следовательно, выражение для метода `Move` в цикле будет таким.

```
Sheets("Лист1").Move Before:= Sheets(1)
```

После выполнения второй итерации цикла выражение имеет следующий вид.

```
Sheets("Лист2").Move Before:= Sheets(2)
```

В процедуру `SortSheets` добавим новый код.

```
Sub SortSheets()
   Dim SheetNames() As String
   Dim SheetCount as Long
   Dim i as Long
   SheetCount = ActiveWorkbook.Sheets.Count
   ReDim SheetNames(1 To SheetCount)
   For i = 1 To SheetCount
      SheetNames(i) = ActiveWorkbook.Sheets(i).Name
   Next i
   Call BubbleSort(SheetNames)
   For i = 1 To SheetCount
      ActiveWorkbook.Sheets(SheetNames(i)).Move _
      Before:=ActiveWorkbook.Sheets(i)
   Next i
End Sub
```

Тестирование показало, что процедура прекрасно работает с рабочей книгой `Test.xlsm`.

Теперь необходимо собрать весь код. Объявим все переменные, используемые в процедурах, и добавим несколько комментариев, а также пустых строк, чтобы программу можно было легче прочесть. В результате процедура `SortSheets` будет приведена к следующему виду.

```
Sub SortSheets()
'  Эта процедура сортирует листы
'  активной рабочей книги по возрастанию.
'  Нажмите клавиши <Ctrl+Shift+S> для выполнения

   Dim SheetNames() As String
   Dim SheetCount As Long
   Dim i As Long

'  Определение количества листов и массива ReDim
   SheetCount = ActiveWorkbook.Sheets.Count
   ReDim SheetNames(1 To SheetCount)

'  Заполнение массива названиями листов
   For i = 1 To SheetCount
```

```
      SheetNames(i) = ActiveWorkbook.Sheets(i).Name
   Next i

'  Сортировка массива по возрастанию
   Call BubbleSort(SheetNames)

'  Перемещение листов
   For i = 1 To SheetCount
      ActiveWorkbook.Sheets(SheetNames(i)).Move _
      Before:= ActiveWorkbook.Sheets(i)
   Next i
End Sub
```

Итак, все работает. Чтобы продолжить проверку программы, добавим еще несколько листов в книгу `Test.xlsm` и изменим некоторые названия. Программа работает прекрасно!

Дополнительное тестирование

Наверное, вы считаете, что работа окончена. Однако тот факт, что процедура работает с рабочей книгой `Test.xlsm`, не означает, что она будет работать со всеми рабочими книгами. Чтобы проверить программу, загрузим несколько других рабочих книг и вновь запустим программу. Скоро вы убедитесь в том, что приложение неидеально (если быть точным, оно далеко от идеала). Были обнаружены следующие проблемы.

- Рабочие книги с большим количеством листов сортируются очень долго, так как при операциях перемещения окно постоянно обновляется.
- Сортировка не всегда выполняется. Например, в одном из тестов лист с названием `SUMMARY` (все буквы в верхнем регистре) был размещен перед листом `Sheet1`. Эта проблема вызвана процедурой `BubbleSort` (так как `U` в верхнем регистре считается больше, чем `h` в нижнем).
- Если на экране не отображаются окна рабочих книг, при нажатии <Ctrl+Shift+S> макрос выдает ошибку.
- Если структура рабочей книги защищена, метод `Move` не работает.
- После сортировки последний лист рабочей книги становится активным. Изменение активного листа — не очень удачное решение проблемы; лучше, если бы активным оставался лист, который был таковым до начала выполнения программы.
- При прерывании макроса с помощью комбинации клавиш <Ctrl+Break> VBA отображает сообщение об ошибке.
- Макрос не может быть "обращен вспять" (вы не можете воспользоваться командой **Отменить** (Undo)). Если пользователь случайно нажмет клавиши <Ctrl+Shift+S>, листы рабочей книги отсортируются, и в исходное состояние придется возвращать их вручную.

Устранение проблем

Решить проблему обновления изображения на экране несложно — для этого вставьте в начале процедуры `SortSheets` такую инструкцию:

```
Application.ScreenUpdating = False
```

Этот оператор "замораживает" окна Excel во время выполнения макроса. Еще один положительный момент заключается в том, что увеличивается скорость выполнения макроса. Когда выполнение макроса завершится, снова включается автоматическое обновление экрана.

Можно было бы исправить и проблему с процедурой `BubbleSort`. Для этого используем функцию `UCase`, чтобы преобразовать названия листов в верхний регистр. Таким образом, все сравнения выполняются с названиями в верхнем регистре. Исправленная строка выглядит следующим образом:

```
If UCase(List(i)) > UCase(List(j)) Then
```

Совет

Еще один способ решения "проблемы регистра" — добавить в начало модуля следующий оператор:

```
Option Compare Text
```

В этом случае VBA выполняет сравнение строк на основе нечувствительных к регистру правил сортировки. Другими словами, `A` считается тем же, что и `a`.

Чтобы избежать сообщения об ошибке, которое появляется, когда все рабочие книги свернуты, добавим процедуру проверки ошибок. Если не существует активных рабочих книг, происходит ошибка. Применим оператор `On Error Resume Next`, чтобы проигнорировать ошибку, и проверим значение `Err`. Если `Err` не равно нулю, это означает, что произошла ошибка. Следовательно, процедура заканчивается. Ниже приведен код проверки ошибок.

```
On Error Resume Next
SheetCount = ActiveWorkbook.Sheets.Count
If Err <> 0 Then Exit Sub ' нет активной рабочей книги
```

Можно и не использовать оператор `On Error Resume Next`. Следующий оператор непосредственно определяет, свернута ли рабочая книга, и не реализует обработку ошибок. Этот оператор находится в верхней части процедуры `SortSheets`.

```
If ActiveWorkbook Is Nothing Then Exit Sub
```

Обычно для защиты структуры рабочей книги имеется серьезная причина. Мы не будем снимать защиту; программа должна отображать предупреждение, чтобы пользователь снял защиту и снова выполнил макрос. Проверку защищенной структуры книги выполнить легко — свойство `ProtectStructure` объекта `WorkBook` возвращает `True`, если книга защищена. Поэтому добавим в проект следующий код.

```
' Проверка защиты структуры рабочей книги
  If ActiveWorkbook.ProtectStructure Then
     MsgBox ActiveWorkbook.Name & " защищена.", _
        vbCritical, "Сортировка листов невозможна."
     Exit Sub
  End If
```

Если структура рабочей книги защищена, пользователь увидит окно сообщения, показанное на рис. 9.11.

Для повторной активизации листа после завершения сортировки я написал код, который сопоставляет исходный лист с объектной переменной (`OldActiveSheet`), а также активизирует этот лист после завершения процедуры. Ниже показан оператор, который инициализирует переменную.

```
Set OldActive = ActiveSheet
```

А следующий оператор активизирует рабочий лист, который был изначально активным:

```
OldActive.Activate
```

После нажатия комбинации клавиш <Ctrl+Break> выполнение макроса обычно приостанавливается, и VBA выдает сообщение об ошибке. Но так как одна из целей проекта — избежать сообщений об ошибке, необходимо вставить команду предотвращения подобной ситуации. В справочной системе указано, что объект `Application` обладает свойством `EnableCancelKey`, которое может отключить комбинацию клавиш <Ctrl+Break>. Поэтому добавим следующий оператор в начало программы:

```
Application.EnableCancelKey = xlDisabled
```

Предупреждение

Будьте очень внимательны, когда отключаете прерывание макроса, выполняемое с помощью клавиш <Ctrl+Break>. Если программа попадет в бесконечный цикл, выйти из него вы не сможете. Лучше использовать этот оператор, когда все работает идеально.

Для предотвращения проблемы, возникающей из-за случайной сортировки листов, перед отключением клавиш <Ctrl+Break> в процедуру был добавлен следующий оператор.

```
If MsgBox("Сортировать листы в активной рабочей книге?", _
   vbQuestion + vbYesNo) <> vbYes Then Exit Sub
```

После вызова на выполнение процедуры `SortSheets` на экране появится сообщение, показанное на рис. 9.12.

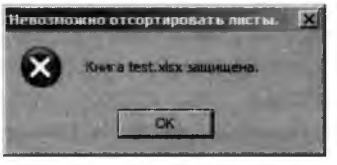


Рис. 9.11. Это сообщение о том, что листы рабочей книги отсортировать нельзя

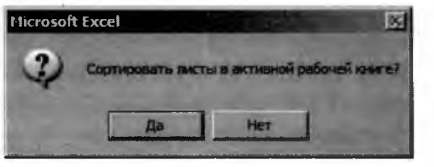


Рис. 9.12. Это сообщение отображается до начала сортировки листов

После выполнения всех описанных выше изменений процедура `SortSheets` принимает следующий вид.

```
Option Explicit
Sub SortSheets()
'  Эта процедура сортирует листы
'  активной рабочей книги в порядке возрастания.
'  Нажмите клавиши <Ctrl+Shift+S> для вызова процедуры

   Dim SheetNames() As String
   Dim i As Long
   Dim SheetCount As Long
   Dim OldActiveSheet As Object

   If ActiveWorkbook Is Nothing Then Exit Sub ' Нет активной рабочей книги
   SheetCount = ActiveWorkbook.Sheets.Count

'  Проверка защищенной структуры рабочей книги
   If ActiveWorkbook.ProtectStructure Then
      MsgBox ActiveWorkbook.Name & " защищена.", _
```

```
        vbCritical, "Листы не могут быть отсортированы."
    Exit Sub
  End If
' Проверка пользователя
  If MsgBox("Сортировать листы в активной рабочей книге?", _
    vbQuestion + vbYesNo) <> vbYes Then Exit Sub
' Отключение комбинации клавиш <Ctrl+Break>
  Application.EnableCancelKey = xlDisabled

' Получение количества листов
  SheetCount = ActiveWorkbook.Sheets.Count

' Изменение размерности массива
  ReDim SheetNames(1 To SheetCount)
' Сохранение ссылки на активный лист
  Set OldActiveSheet = ActiveSheet

' Заполнение массива названиями листов
  For i = 1 To SheetCount
    SheetNames(i) = ActiveWorkbook.Sheets(i).Name
  Next i

' Сортировка массива в порядке возрастания
  Call BubbleSort(SheetNames)

' Отключение обновления экрана
  Application.ScreenUpdating = False

' Перемещение листов
  For i = 1 To SheetCount
    ActiveWorkbook.Sheets(SheetNames(i)).Move _
      Before:=ActiveWorkbook.Sheets(i)
  Next i
' Повторная активизация исходного активного листа
  OldActiveSheet.Activate
End Sub
```

Доступность

Макрос `SortSheets` сохранен в личной книге макросов, поэтому он всегда доступен при запуске Excel. На этом этапе макрос может выполняться при выборе названия макроса в диалоговом окне **Макрос** (Macro) — это окно отображается после нажатия клавиш <Alt+F8> или <Ctrl+Shift+S>. Команду вызова этого макроса можно также добавить на ленту.

Для добавления команды на ленту выполните следующие действия.

1. Щелкните правой кнопкой мыши на ленте и в контекстном меню выберите команду **Настройка ленты** (Customize the Ribbon).
2. На вкладке **Настройка ленты** (Customize Ribbon) диалогового окна **Параметры Excel** (Excel Options) в списке **Выбрать команды** (Choose Commands From) выберите категорию **Макросы** (Macros).
3. Щелкните на значке `PERSONAL.XLSB!SortSheets`.
4. Используйте элементы управления в правом окне для создания новой вкладки и группы ленты. (Вы не сможете добавить команду в существующую группу.)

Я создал группу **Листы** (Worksheets) во вкладке **Вид** (View) и переименовал новый, добавленный в эту группу элемент, на **Сортировка листов** (Short Sheets) (рис. 9.13).

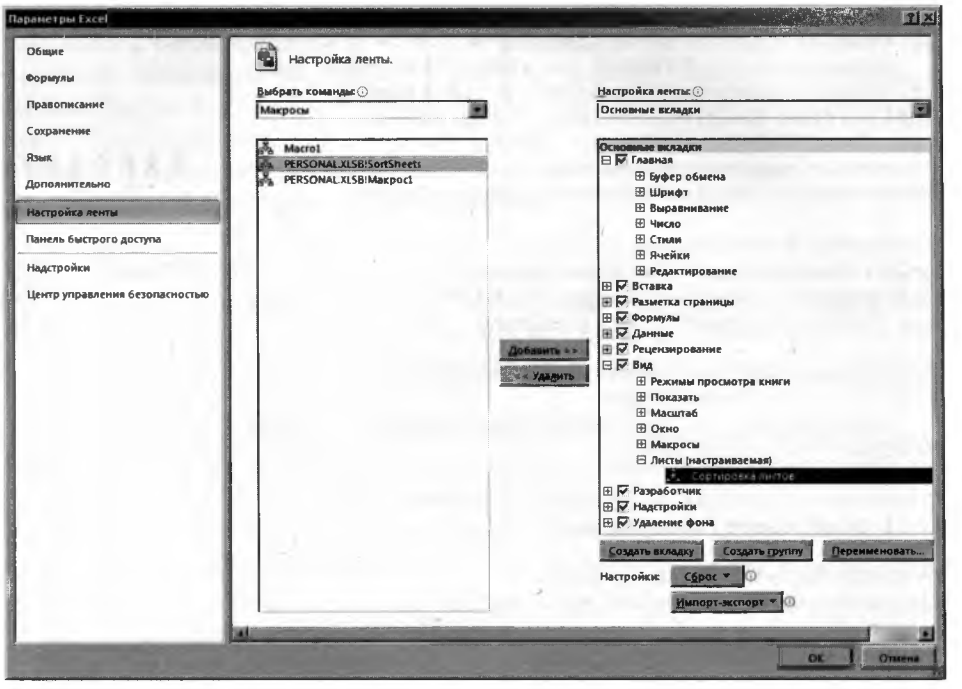


Рис. 9.13. Добавление новой команды на ленту

Оценка проекта

Итак, результат получен. Утилита соответствует всем изначальным требованиям: она сортирует все листы в активной рабочей книге, ее можно легко выполнить, она всегда доступна, она выполняется (что легко проверить) для всех рабочих книг и пока еще не отображала сообщений об ошибке VBA.

Примечание

В процедуре все еще присутствует одна небольшая проблема: сортировка достаточно строгая и не всегда кажется "логичной". Например, после сортировки лист `Лист10` размещается перед листом `Лист2`. Большинство пользователей предпочитают видеть лист `Лист2` перед листом `Лист10`. Решить эту проблему довольно сложно и это выходит за рамки учебных примеров, рассматриваемых в данной книге.

Глава 10

Создание функций

В этой главе...

- Процедуры и функции
- Назначение пользовательских функций
- Простой пример функции
- Синтаксис функции
- Аргументы функций
- Примеры функций
- Имитация функции СУММ
- Расширенные функции для работы с датами
- Отладка функций
- Работа с диалоговым окном Мастер функций
- Использование надстроек для хранения пользовательских функций
- Использование функций Windows API

А теперь приступим к важнейшей теме, без рассмотрения которой невозможно двигаться дальше, — к функциям VBA.

Процедуры и функции

Функция — это процедура VBA, которая выполняет вычисления и возвращает значение. Функции можно использовать в коде Visual Basic for Applications (VBA) или в формулах.

Процедуру можно рассматривать как команду, которая выполняется пользователем или другой процедурой. С другой стороны, процедуры типа `Function` обычно возвращают отдельное значение (или массив) подобно функциям рабочих листов Excel и встроенным функциям VBA. Как и встроенные функции, процедуры типа `Function` имеют аргументы.

Процедуры типа `Function` универсальны и используются в двух ситуациях:

- как часть выражения в процедуре VBA;
- в формулах, которые создаются на рабочем листе.

Процедуру типа `Function` можно использовать в работе с функциями Excel и встроенными функциями VBA. Единственное исключение — невозможно использовать функцию VBA в формуле проверки данных.

Перекрестная ссылка

В главе 11 представлены полезные и часто используемые примеры функций. Вы можете воспользоваться многими из них в своей работе.

Назначение пользовательских функций

Несомненно, вам знакомы функции Excel. Даже начинающие пользователи знают, как работать с самыми популярными функциями в формулах рабочего листа — СУММ, СРЗНАЧ и ЕСЛИ. Excel 2010 содержит более 400 встроенных функций. Если этого количества недостаточно, можно создавать пользовательские функции с помощью VBA.

Зная о существовании большого количества функций, доступных в Excel и VBA, вы можете заняться созданием новых функций. Некоторые из них смогут упростить жизнь рядовым пользователям. Тщательно спланированные специальные функции эффективно используются в формулах на рабочем листе и в процедурах VBA.

Например, зачастую пользовательские функции создаются, чтобы сократить формулы. Короткие формулы намного легче воспринимаются и обрабатываются. Однако следует отметить, что специальные функции, используемые в формулах, обычно выполняются медленнее, чем встроенные.

При создании приложений обратите внимание на то, что в некоторых процедурах часто повторяются одни и те же вычисления. В таком случае стоит подумать о создании специальной функции, выполняющей эти вычисления, и впоследствии ее достаточно будет лишь вызвать из процедуры. Специальная функция, например, поможет заменить повторяющийся код программы и сократить количество возможных ошибок.

Созданные вами функции окажутся полезными и для ваших коллег. Другие программисты, возможно, захотят приобрести специальные функции, которые экономят рабочее время и затрачиваемые на их программирование усилия.

Вас может пугать мысль о создании пользовательских функций, однако этот процесс несложен. Например, я люблю создавать пользовательские функции, они отображаются в диалоговом окне Мастер функций (Insert Function) наряду со встроенными функциями Excel. Возникает ощущение, будто вы самостоятельно обновляете программное обеспечение.

В этой главе вы узнаете об этапах создания пользовательских функций, причем теория будет подкреплена полезными примерами.

Простой пример функции

Чтобы сразу же перейти к делу, рассмотрим пример функции VBA. Ниже приведена пользовательская функция, определенная в модуле VBA. Она называется `RemoveVowels` и принимает один аргумент. Затем она удаляет все гласные буквы и возвращает аргумент.

```
Function RemoveVowels(Txt) As String
'  Удаляет все гласные буквы из текстового аргумента
```

```
   Dim i As Long
   RemoveVowels = ""
   For i = 1 To len(Txt)
      If Not UCase(mid(Txt, i, 1)) Like "[АЕIOUAЕИОУЫЭЮЯ]" Then
         RemoveVowels = RemoveVowels & mid(Txt, i, 1)
      End If
   Next i
End Function
```

Конечно, эта не самая полезная функция из созданных мною, но она демонстрирует некоторые ключевые концепции, связанные с функциями. Принцип ее работы будет объяснен в разделе "Анализ пользовательской функции".

Предупреждение

Создавая пользовательские функции, которые используются в формуле рабочего листа, убедитесь, что код вводится в обычном модуле VBA. Если вы поместите пользовательские функции в модуле листа `Лист` (Sheet), в пользовательской форме (UserForm) или в модуле `ЭтаКнига` (ThisWorkbook), они не будут выполняться в формулах.

Использование функции на рабочем листе

При вводе формулы, в которой используется функция `RemoveVowels`, Excel выполняет программу для получения конечного значения. Эту функцию можно использовать в следующей формуле:

```
=RemoveVowels(A1)
```

Примеры данной функции в действии показаны на рис. 10.1. Формулы введены в столбце B, в качестве аргумента используется текст в столбце A. Функция возвращает свой аргумент, в котором удалены гласные буквы.

	A	B
1	Хорошие девочки все делают хорошо.	Хрш двчк вс длт хрш.
2	абвгдежзиклам	бвгджзклм
3	Microsoft Excel	Mcrsft xcl
4	abcdijklmnoprst	bcdjklmnprst
5	Сбой связи.	Сбй свз.
6	Из этого предложения удаляются гласные буквы.	з тг прдлжн длтс глсн бкв.
7		
8		

Рис. 10.1. Применение пользовательской функции в формуле рабочего листа

Фактически такая функция действует подобно любой встроенной функции рабочего листа. Вы можете вставить функцию в формулу, используя команду **Формулы⇨Библиотека функций⇨Вставить функцию** (Formulas⇨Function Library⇨Insert Function) или кнопку мастера функций в левой части строки формул. Любое из этих действий приведет к появлению диалогового окна **Мастер функций** (Insert Function), в котором находятся пользовательские функции (обычно — в категории **Определенные пользователем** (User Defined)).

Вы также можете создавать вложенные пользовательские функции и сочетать их в формулах с другими элементами. Например, приведенная ниже формула вкладывает функцию `RemoveVowels` в функцию Excel `ПРОПИСН` (Upper). В результате исходная строка преобразуется в символы верхнего регистра (без гласных).

```
=ПРОПИСН(RemoveVowels(A1))
```

Использование функции в процедуре VBA

Пользовательские функции можно применять не только в формулах рабочего листа, но и в процедурах VBA. Следующая процедура VBA, определенная в том же модуле, что и пользовательская функция `RemoveVowels`, сначала отображает окно для ввода текста пользователем. Затем процедура применяет встроенную функцию VBA `MsgBox` для отображения данных, введенных пользователем, но уже после их обработки функцией `RemoveVowels` (рис. 10.2). Первоначальные данные отображаются в заголовке окна сообщения.

```
Sub ZapTheVowels()
   Dim UserInput as String
   UserInput = InputBox("Введите текст:")
   MsgBox RemoveVowels(UserInput), , UserInput
End Sub
```

В примере, показанном на рис. 10.2, в ответ на приглашение функции `InputBox` была введена строка `Профессиональное программирование на VBA в Excel`. Функция `MsgBox` отображает текст без гласных букв.

Рис. 10.2. Применение пользовательской функции в процедуре VBA

Чего нельзя сделать с помощью пользовательских функций

При разработке пользовательских функций важно понимать различия между функциями, которые вызываются из других процедур VBA, и функциями, используемыми в формулах рабочего листа. Функции, используемые в формулах рабочего листа, — "пассивные". Например, такая функция не может обрабатывать диапазоны либо изменять содержимое рабочего листа. Соответствующий практический пример приведен ниже.

Можете попробовать написать функцию рабочего листа, которая изменяет содержимое ячейки. Например, рекомендуется всегда иметь под рукой формулу, которая изменяет цвет текста в ячейке в зависимости от значения этой ячейки. Однако несмотря ни на что, такую функцию написать невозможно. Что бы вы ни делали, функция всегда будет возвращать ошибку. Помните: функция возвращает значение, но не может выполнять операции над объектами.

Следует отметить, что из этого правила имеется одно исключение. Вы можете изменить текст комментария ячейки с помощью пользовательской функции VBA, код которой приведен ниже.

```
Function ModifyComment(Cell As Range, Cmt As String)
   Cell.Comment.Text Cmt
End Function
```

Далее приведен пример использования этой функции в формуле. Текст комментария в ячейке A1 заменяется новым текстом. Функция не работает, если в ячейке A1 отсутствует комментарий.

```
=ModifyComment(A1,"Комментарий был изменен")
```

Анализ пользовательской функции

Функции могут быть достаточно сложными, если это необходимо. Как правило, они более сложны и намного более полезны, чем процедура в приведенном ниже примере. И анализ рассматриваемого в предыдущем разделе примера функции поможет вам разобраться в данном вопросе.

Приведем текст функции.

```
Function RemoveVowels(Txt) As String
' Удаляет все гласные буквы из текстового аргумента
  Dim i As Long
  RemoveVowels = ""
  For i = 1 To Len(Txt)
     If Not UCase(Mid(Txt, i, 1)) Like _
     "[АЕIOUAЕИОУЫЭЮЯ]" Then
        RemoveVowels = RemoveVowels & Mid(Txt, i, 1)
     End If
  Next i
End Function
```

Обратите внимание, что функция начинается с ключевого слова `Function`, а не `Sub`, после которого указывается название функции (`RemoveVowels`). Эта специальная функция использует только один аргумент (`Txt`), заключенный в скобки. Ключевое слово `As String` определяет тип данных значения, которое возвращает функция. (Excel по умолчанию использует тип данных `Variant`, если тип данных не определен.)

Вторая строка — простой комментарий (необязательный), который описывает выполняемые функцией действия. После комментария приведен оператор `Dim`, который объявляет переменную (`i`), применяемую в функции. Тип этой переменной — `Long`.

Примечание

Обратите внимание, что в данном случае в качестве переменной используется имя функции. Как только функция завершает свое выполнение, возвращается текущее значение переменной, которое соответствует названию функции.

Следующие пять инструкций образуют цикл `For-Next`. Процедура циклически просматривает каждый символ введенного текста, создавая на их основе строку. Первая инструкция в цикле использует функцию VBA `Mid`, которая возвращает единственный символ строки ввода, а также преобразует этот символ в символ верхнего регистра. Затем этот символ сравнивается со списком символов с помощью оператора Excel `Like`. Другими словами, значение выражения `If` будет `True`, если символ отличен от символов A, E, I, O, U, А, Е, И, О, У, Ы, Э, Ю и Я. В подобных случаях символ добавляется к переменной `RemoveVowels`.

По завершении цикла из строки ввода удаляются все гласные буквы. Эта строка и является значением, возвращаемым функцией `RemoveVowels`. Сама процедура завершается оператором `End Function`.

Имейте в виду, что кодирование функции может осуществляться различными способами. Ниже приведен код функции, которая выполняет те же действия, что и предыдущая функция, но закодирована иначе.

```
Function RemoveVowels(txt) As String
' Удаляет все гласные буквы из текстового аргумента (Txt)
  Dim i As Long
  Dim TempString As String
```

```
    TempString = ""
    For i = 1 To Len(txt)
       Select Case ucase(Mid(txt, i, 1))
          Case "A", "E", "I", "O", "U", "А", "Е", "И", _
          "О", "У", "Ы", "Э", "Ю", "Я"
          '  Ничего не делать
          Case Else
             TempString = TempString & Mid(txt, i, 1)
          End Select
       Next i
    RemoveVowels = TempString
End Function
```

В этой версии программы использовалась текстовая переменная (`TempString`), в которой хранится конструируемая строка символов (без гласных букв). Затем, перед завершением процедуры, содержимое переменной `TempString` было присвоено названию функции. В этой версии также применяется конструкция `Select Case` вместо конструкции `If-Then`.

Компакт-диск

Обе версии описываемой в этом разделе функции можно найти на прилагаемом к книге компакт-диске в файле `remove vowels.xlsm`.

Синтаксис функции

Пользовательские функции имеют много общего с процедурами типа `Sub`. (Детально о процедурах типа `Sub` рассказано в главе 9.)

Для объявления функции применяется следующий синтаксис.

```
[Public | Private][Static] Function имя ([список_аргументов])[As тип]
   [инструкции]
   [имя = выражение]
   [Exit Function]
   [инструкции]
   [имя = выражение]
End Function
```

Функция состоит из следующих ключевых элементов.

- `Public` (необязательное ключевое слово). Указывает, что функция доступна для других процедур во всех остальных модулях активных проектов VBA.
- `Private` (необязательное ключевое слово). Указывает, что функция доступна только для других процедур в текущем модуле.
- `Static` (необязательное ключевое слово). Указывает, что значения переменных, которые объявлены в функции, сохраняются между вызовами функции.
- `Function` (обязательное ключевое слово). Обозначает начало функции, возвращающей значение или другие данные.
- *Имя* (обязательное ключевое слово). Представляет произвольное название функции. При именовании функций используются те же правила, что и при задании имен переменным. По окончании выполнения функции результат присваивается ее названию.

- *Список_аргументов* (необязательный). Список одной или нескольких переменных, представляющих аргументы, которые передаются в функцию. Аргументы заключены в скобки. Для разделения пар аргументов используется запятая.
- *Тип* (необязательный). Тип данных, который возвращает функция.
- *Инструкции* (необязательные). Корректные инструкции VBA.
- `Exit Function` (необязательный). Оператор, вызывающий немедленный выход из функции до ее завершения.
- `End Function` (обязательное). Ключевое слово, обозначающее конец функции.

О пользовательских функциях, написанных на VBA, необходимо знать следующее: значение всегда присваивается названию функции минимум один раз и, как правило, тогда, когда функция завершила выполнение.

Создание пользовательской функции начните с создания модуля VBA (можно также использовать существующий модуль). Введите ключевое слово `Function`, после которого укажите название функции и список ее аргументов (если они есть) в скобках. Вы также можете объявить тип данных значения, которое возвращает функция, используя ключевое слово `As` (это делать необязательно, но рекомендуется). Вставьте код VBA, выполняющий требуемые действия, и убедитесь, что необходимое значение присваивается переменной процедуры, соответствующей названию функции, минимум один раз в теле функции. Функция заканчивается оператором `End Function`.

Имена функций подчиняются тем же правилам, что и имена переменных. Если вы планируете использовать функцию в формуле рабочего листа, убедитесь, что название не имеет форму адреса ячейки. Например, если в качестве имени функции указывается `J21`, заключите его в кавычки: `='J21'(A1)`.

Лучше всего вообще не использовать имена функций, являющиеся ссылками на ячейки, включая именованные диапазоны. Также не присваивайте функциям имена, которые соответствуют названиям встроенных функций Excel. В случае возникновения конфликта между именами функций программа Excel всегда делает выбор в пользу встроенных функций.

Область действия функции

В главе 9 была рассмотрена концепция области действия процедуры (`Public` или `Private`). То же самое относится и к функциям: область действия функции определяет, может ли она быть вызвана процедурами в других модулях или рабочих листах.

Ниже представлены условия, о которых следует помнить при определении области действия функции.

- Если область действия функции не задана, то по умолчанию подразумевается `Public`.
- Функции, объявленные как `Private`, не отображаются в диалоговом окне **Мастер функций** (Insert Function). Следовательно, при создании функции, которая должна использоваться только в процедуре VBA, необходимо объявить ее как `Private`, чтобы пользователи не пытались применять ее в формуле.
- Если в программе VBA необходимо вызвать функцию, которая определена в другой рабочей книге, задайте ссылку на другую рабочую книгу, воспользовавшись командой VBE **Tools⇨References** (Сервис⇨Ссылки).

Выполнение функций

В то время как процедура (Sub) может вызываться на выполнение различными способами, функцию (Function) можно вызвать одним из следующих способов:

- вызвать ее из другой процедуры;
- включить ее в формулу рабочего листа;
- включить в формулу условного форматирования;
- вызвать ее в окне отладки VBE (Immediate).

Вызов из процедуры

Пользовательские функции можно вызывать из процедуры точно так же, как и встроенные функции. Например, после определения функции с названием SumArray в код вводится оператор, показанный ниже.

```
Total = SumArray(MyArray)
```

Этот оператор выполняет функцию SumArray с аргументом MyArray, возвращает результат функции и присваивает его переменной Total.

Кроме того, можно использовать метод Run объекта Application. Соответствующий пример приводится ниже.

```
Total = Application.Run ("SumArray", "MyArray")
```

Первый аргумент метода Run — имя функции. Дополнительные аргументы представляют аргумент (аргументы) функции. Аргументами метода Run могут быть строки (как показано выше), числа или переменные.

Использование в формуле рабочего листа

Применение специальных функций в формуле рабочего листа во многом подобно применению встроенных функций, за некоторым исключением. Удостоверьтесь, что Excel может найти необходимую функцию. Если функция сохранена в той же рабочей книге, то ничего особенного делать не нужно. Если же функция находится в другой рабочей книге, то необходимо указать Excel, где ее найти.

Вы вправе выбрать один из трех способов.

- **Указать перед названием функции ссылку на файл.** Например, если вы решили использовать функцию с названием CountNames, определенную в открытой рабочей книге Myfunc.xlsm, то обратитесь к следующей ссылке:

  ```
  =Myfuncs.xlsm!CountNames(A1:A1000)
  ```

  Если вы вставите функцию в диалоговом окне мастера функций (Insert Function), то ссылка на рабочую книгу будет добавлена автоматически.

- **Установить ссылку на рабочую книгу.** Это можно сделать с помощью команды VBE Tools⇨References (Сервис⇨Ссылки). Если функция определена в рабочей книге, на которую установлена ссылка, то не следует указывать имя рабочего листа. Даже если зависимая рабочая книга определена по ссылке, в диалоговом окне Мастер функций (Insert Function) указывается рабочая книга, содержащая функцию (хотя и необязательно).

- **Создать надстройку.** При создании надстройки на основе рабочей книги, в которой представлены функции, можно не задавать ссылку на файл, если одна из функций используется в формуле. Однако надстройку необходимо устанавливать. Детально надстройки рассматриваются в главе 21.

В отличие от процедур, функции не отображаются в диалоговом окне Макрос (Macro) при выполнении команды Разработчик⇨Код⇨Макросы (Developer⇨Code⇨Macros). Кроме того, функцию нельзя выбрать при использовании команды VBE Run⇨Sub/UserForm (Выполнить⇨Процедуру/пользовательскую форму) (или нажатии <F5>), если курсор установлен в тексте функции (так как при этом отображается диалоговое окно Макрос, в котором можно выбрать макрос для выполнения). Именно поэтому необходимо выполнить дополнительные действия по тестированию функций еще в процессе разработки. Один из возможных подходов — создать простую процедуру, вызывающую функцию. Если функция должна использоваться в формулах рабочего листа, то для ее проверки следует ввести простую формулу.

Использование в формулах условного форматирования

Условное форматирование подразумевает создание формул, которые являются логическими (т.е. возвращают значения Истина (True) или Ложь (False)). Если формула возвращает значение Истина, условие выполняется, и форматирование применяется к выбранной ячейке.

В формулах, определяющих условное форматирование, могут применяться пользовательские функции VBA. Например, ниже приводится пример простой функции VBA, которая возвращает значение Истина в том случае, если в качестве ее аргумента выступает ячейка, которая включает формулу.

```
Function CELLHASFORMULA(cell) As Boolean
  CELLHASFORMULA = cell.HasFormula
End Function
```

После определения этой функции в модуле VBA можно установить правило условного форматирования, суть которого заключается в том, что ячейки с формулами будут отформатированы особым образом.

1. Выделите диапазон ячеек, к которому будет применяться условное форматирование.

   Например, выделите ячейки A1:G20.

2. Выполните команду Главная⇨Стили⇨Условное форматирование⇨Создать правило (Home⇨Styles⇨Conditional Formatting⇨New Rule).

3. В диалоговом окне Создание правила форматирования (New Formatting Rule) выберите параметр Использовать формулу для определения форматируемых ячеек (Use a Formula to Determine Which Cells to Format).

4. В поле ввода введите формулу, при этом следите, чтобы аргумент, определяющий ячейку, соответствовал ячейке, находящейся в верхнем левом углу диапазона, который был выделен в п. 1.

   ```
   =CELLHASFORMULA(A1)
   ```

5. Щелкните на кнопке Формат (Format) для определения формата ячеек, которые соответствуют условию форматирования.

6. Щелкните на кнопке OK для применения правила условного форматирования к выделенному диапазону.

Ячейки из диапазона, который содержит формулу, будут отображаться с применением выбранного вами формата. На рис. 10.3 показано диалоговое окно Создание правила форматирования, в котором для определения формулы применяется пользовательская функция.

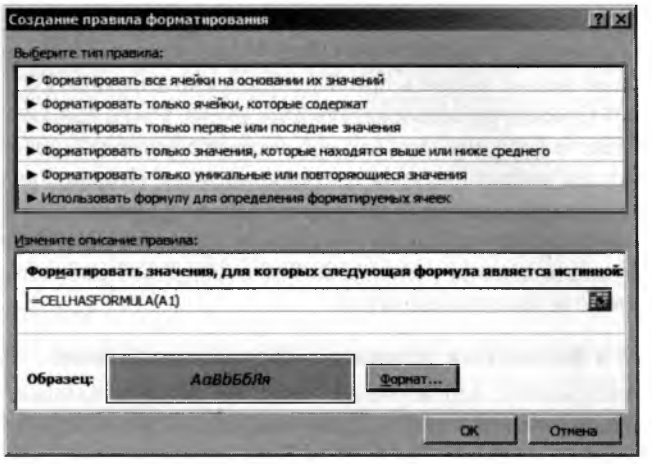


Рис. 10.3. Применение пользовательской функции VBA при условном форматировании

Вызов функции из окна отладки в VBE

Еще один способ вызова функции на выполнение — обратиться к окну отладки VBE (Immediate). Этот метод обычно применяется на этапе тестирования. На рис. 10.4 показан соответствующий пример.

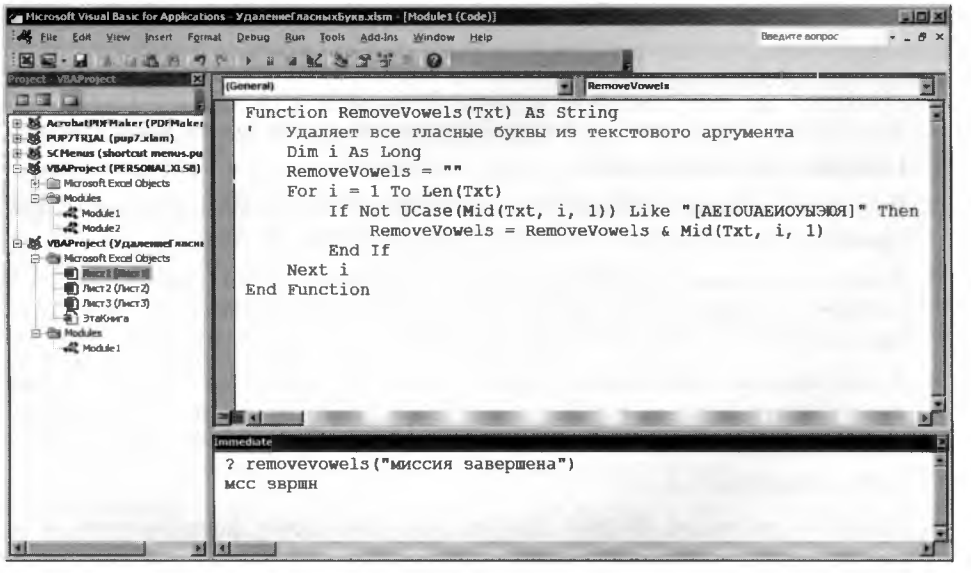


Рис. 10.4. Вызов функции в окне отладки

Изобретаем колесо

Развлечения ради я написал собственную версию функции Excel ПРОПИСН (преобразует символы строки в верхний регистр) и назвал ее UpCase.

```
Function UpCase(InString As String) As String
'  Преобразует символы аргумента в верхний регистр.
   Dim StringLength As Integer
```

```
    Dim i As Integer
    Dim ASCIIVal As Integer
    Dim CharVal As Integer

    StringLength = Len(InString)
    UpCase = InString
    For i = 1 To StringLength
       ASCIIVal = Asc(Mid(InString, i, 1))
       CharVal = 0
       If ASCIIVal >= 97 And ASCIIVal <= 122 Then
          CharVal = -32
          Mid(UpCase, i, 1) = Chr(ASCIIVal + CharVal)
       End If
    Next i
End Function
```

Примечание. Рабочая книга, содержащая эту функцию, находится на прилагаемом к книге компакт-диске в файле `upper case.xlsm`.

Обратите внимание, что я выбрал относительно простой путь, — обратился к функции VBA `UCase`.

Итак, мне было интересно посмотреть, чем пользовательская функция отличается от встроенной, поэтому я создал рабочий лист, который вызывает функцию 20 тысяч раз с помощью случайных имен. Вычисления продолжались примерно 20 секунд. Затем я подставил встроенную функцию Excel `UPPER` и выполнил тест еще раз. На этот раз вычисления были выполнены практически мгновенно. Конечно, моя функция `UpCase` вряд ли работает на основе оптимального алгоритма, более того, любая встроенная функция Excel работает быстрее самой лучшей пользовательской функции.

Еще один пример "изобретения колеса" можно найти в разделе "Имитация функции `СУММ`".

Аргументы функций

Всегда помните о следующих особенностях аргументов процедур-функций:

- аргументы могут представляться переменными (в том числе массивами), константами, символьными данными или выражениями;
- некоторые функции не имеют аргументов;
- определенные функции имеют строго заданное число обязательных аргументов (от 1 до 60);
- отдельные функции имеют как обязательные, так и необязательные аргументы.

Примечание

Если формула содержит пользовательскую функцию рабочего листа и возвращает `#ЗНАЧ!`, это означает наличие ошибки в функции. Ошибка может быть вызвана логическими ошибками в программе или передачей в функцию некорректных аргументов (см. раздел "Отладка функций").

Примеры функций

В настоящем разделе приведен ряд примеров, иллюстрирующих эффективное использование аргументов функций. Данный материал касается также процедур (`Sub`).

Функции без аргументов

Как и процедуры, пользовательские функции необязательно должны иметь аргументы. Например, в Excel есть несколько таких встроенных функций, в том числе СЛЧИС, СЕГОДНЯ и ТДАТА. Несложно создать аналогичные функции.

Ниже вы найдете примеры функций, в которых аргументы не используются.

Компакт-диск

Рабочая книга, содержащая функции без аргументов, находится на прилагаемом к книге компакт-диске. Соответствующий файл называется no argument.xlsm.

Ниже приведен пример простой функции, не использующей аргументов. Следующая функция возвращает свойство UserName объекта Application. Это имя отображается в диалоговом окне Excel Параметры Excel (Excel Options), раздел Основные (General), и хранится в системном реестре Windows.

```
Function User()
'   Возвращает имя пользователя
    User = Application.UserName
End Function
```

При вводе следующей формулы ячейка возвращает имя текущего пользователя (предполагается, что оно сохранено в реестре).

```
=User()
```

Примечание

При использовании функции без аргумента в формуле рабочего листа необходимо указать пустые скобки. Это требование необязательно, если функция вызывается в процедуре VBA, но включение пустых скобок означает, что вызывается именно функция.

Чтобы использовать данную функцию в другой процедуре, ее следует присвоить переменной, применить в выражении или использовать как аргумент другой функции.

В следующем примере вызывается функция User и в качестве аргумента оператора MsgBox используется значение, которое она возвращает. Оператор конкатенации (&) объединяет текстовую строку с результатом функции User.

```
Sub ShowUser()
    MsgBox "Ваше имя " & User()
End Sub
```

В следующем примере демонстрируется, каким образом можно создать функцию-оболочку, которая возвращает свойство или результат выполнения функции VBA. Ниже приведены три дополнительные функции-оболочки, которые не имеют аргументов.

```
Function ExcelDir() As String
'   Возвращает название папки, в которой установлена Excel
    ExcelDir = Application.Path
End Function

Function SheetCount()
'   Возвращает количество листов в рабочей книге
    SheetCount = Application.Caller.Parent.Parent.Sheets.Count
End Function
```

```
Function SheetName()
' Возвращает имя рабочего листа
  SheetName = Application.Caller.Parent.Name
End Function
```

Управление пересчетом функций

Возможно, вам будет интересно узнать, когда при использовании пользовательской функции в формуле рабочего листа пересчитывается ее значение.

Пользовательские функции ведут себя подобно встроенным функциям Excel. Обычно пользовательская функция пересчитывается тогда, когда это нужно, т.е. в случае изменения одного из аргументов функции. Однако вы можете выполнять пересчет функций чаще. Функция пересчитывается при изменении любой ячейки, если в процедуру добавлен указанный ниже оператор. В процессе использования режима автоматических вычислений они выполняются при изменении любой ячейки.

```
Application.Volatile True
```

Метод `Volatile` объекта `Application` имеет один аргумент (`True` или `False`). Если функция выделена как `volatile` (изменяемая), она пересчитывается всякий раз, когда изменяется любая ячейка листа.

Например, поведение пользовательской функции `StaticRand` может быть изменено с помощью метода `Volatile` так, чтобы оно напоминало поведение функции Excel СЛЧИС.

```
Function NonStaticRand()
' Возвращает случайное число,
' изменяется при каждом пересчете
  Application.Volatile True
  NonStaticRand = Rnd()
End Function
```

При использовании аргумента `False` метода `Volatile` функция пересчитывается только тогда, когда в результате пересчета изменяется один из ее аргументов. (Если функция не имеет аргументов, этот метод неэффективен.)

Чтобы выполнить полный пересчет формул, включая неизменные пользовательские функции, нажмите клавиши <Ctrl+Alt+F9>. Эта комбинация клавиш, к примеру, генерирует новые случайные числа с помощью описанной в этой главе функции `StaticRand`.

А теперь рассмотрим еще один пример функции без аргументов. Для быстрого заполнения диапазона ячеек используется функция Excel СЛЧИС. Но мне не слишком понравилось, что случайные числа изменяются при каждом пересчете рабочего листа. Поэтому я преобразовал формулы в их значения.

Затем я разработал функцию, которая возвращает случайные числа, не изменяющиеся при пересчете формул. Для этого была использована встроенная функция VBA Rnd, которая возвращает случайное число в диапазоне от 0 до 1. Эта функция будет выглядеть следующим образом.

```
Function StaticRand()
' Возвращает случайное число, которое
' не изменяется при пересчете формул
  StaticRand = Rnd()
End Function
```

Если нужно сгенерировать набор случайных чисел между 0 и 1000, воспользуйтесь следующей формулой:

```
=INT(StaticRand()*1000)
```

Значения, полученные с помощью этой формулы, никогда не изменяются. Но у пользователя остается возможность пересчета формулы с помощью комбинации клавиш <Ctrl+Alt+F9>.

Функция с одним аргументом

В этом разделе описана функция, используемая для подсчета комиссионных с объема продаж одного из менеджеров. Вычисления основываются на следующей таблице значений.

Объем продаж за месяц ($)	Ставка комиссионных (%)
0–9999	8,0
10000–19999	10,5
20000–39999	12,0
Больше 40000	14,0

Обратите внимание, что ставка комиссионных изменяется не линейно и зависит от общего объема продаж за месяц. Служащие, которые продают больше, получают более высокие комиссионные.

Существует несколько способов вычислить комиссионные для различных объемов продаж, введенных в таблицу Excel. Потратив немного времени, вы можете получить длинную формулу, подобную приведенной ниже.

```
=ЕСЛИ(И(A1>=0,A1<=9999.99),A1*0.08,
ЕСЛИ(И(A1>=10000,A1<=19999.99),A1*0.105,
ЕСЛИ(И(A1>=20000,A1<=39999.99),A1*0.12,
ЕСЛИ(A1>=40000,A1*0.14,0))))
```

Такая идея неудачна по нескольким причинам. Во-первых, формула чрезмерно сложна, и ее нелегко понять. Во-вторых, значения строго определены в формуле, из-за чего ее сложно изменять.

Рекомендуется применить описанный ниже подход (не требующий программирования на VBA): использовать для вычисления ставки комиссионных функцию просмотра таблицы соответствия ВПР (VLOOKUP). Например, следующая формула использует функцию ВПР для выборки значения комиссионных из диапазона `Table`, а также умножает его на величину, которая хранится в ячейке A1.

```
=ВПР(A1,Table,2)*A1
```

Еще лучше (тогда не нужно использовать таблицу соответствия) создать пользовательскую функцию, представленную ниже.

```
Function Commission(Sales)
   Const Tier1 = 0.08
   Const Tier2 = 0.105
   Const Tier3 = 0.12
   Const Tier4 = 0.14
'  Вычисляет комиссионные с объема продаж
   Select Case Sales
      Case 0 To 9999.99: Commission = Sales * Tier1
      Case 1000 To 19999.99: Commission = Sales * Tier2
      Case 20000 To 39999.99: Commission = Sales * Tier3
```

```
    Case Is >= 40000: Commission = Sales * Tier4
  End Select
End Function
```

После ввода в модуль VBA эту функцию можно использовать в формуле на рабочем листе или вызвать из других процедур VBA.

При вводе в ячейку следующей формулы будет получен результат 3000 (объем продаж 25000 соответствует ставке комиссионных 12%).

```
=Commission(25000)
```

Даже если на рабочем листе не требуется применять пользовательские функции, их создание значительно упрощает программирование на VBA. Например, если процедура VBA вычисляет комиссионные с продаж, то можно использовать точно такую же функцию и вызвать ее из процедуры VBA. Ниже приведена небольшая процедура, запрашивающая у пользователя объем продаж и использующая функцию `Commission` для вычисления ставки комиссионных.

```
Sub CalcComm()
  Dim Sales as Long
  Sales = InputBox("Введите объем продаж:")
  MsgBox "Комиссионные составляют " & Commission(Sales)
End Sub
```

Процедура `CalcComm` сначала отображает окно ввода данных и запрашивает у пользователя объем продаж. Затем она отображает окно сообщения с вычисленными комиссионными для введенного объема продаж.

Данная процедура выполняется, но она далека от совершенства. Ниже показана усовершенствованная версия процедуры, отображающая отформатированные значения и предлагающая повторить цикл вычислений, пока пользователь не щелкнет на кнопке **Нет** (рис. 10.5).

```
Sub CalcComm()
  Dim Sales As Long
  Dim Msg As String, Ans As String

' Запрос о вводе объема продаж
  Sales = Val(InputBox("Введите объем
продаж:", _
     "Калькулятор комиссионных с продаж "))

' Создание сообщения
  Msg = "Объем продаж:" & vbTab
        & Format(Sales, "$#,##0.00")
  Msg = Msg & vbCrLf & "Комиссионные:" & vbTab
  Msg = Msg & Format(Commission(Sales), "$#,##0.00")
  Msg = Msg & vbCrLf & vbCrLf & "Новое вычисление?"

' Отображение результата и запрос следующего вычисления
  Ans = MsgBox(Msg, vbYesNo, "Калькулятор комиссионных с продаж")
  If Ans = vbYes Then CalcComm
End Sub
```

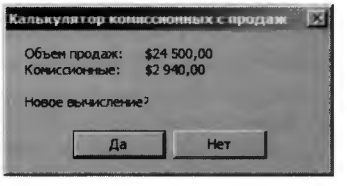


Рис. 10.5. Использование функции для отображения результатов вычисления комиссионных с продаж

Эта функция использует две встроенные константы VBA: `vbTab` обозначает табуляцию (чтобы отделить результат), а `vbCrLf` определяет возврат каретки (для перехода на следующую строку). Функция VBA `Format` указывает значение в заданном формате (в данном случае со знаком доллара, пробелом для разделения разрядов и двумя десятичными знаками).

В обоих этих примерах функция Commission должна содержаться в активной рабочей книге; в противном случае Excel отобразит сообщение об ошибке, указывающее, что функция не определена.

Используйте аргументы, а не ссылки на ячейки

Все применяемые в пользовательской функции диапазоны должны передаваться в качестве аргументов. Рассмотрим функцию, которая возвращает значение в ячейке A1, умноженное на 2.

```
Function DoubleCell()
   DoubleCell = Range("A1") * 2
End Function
```

Хотя эта функция работает, в некоторых случаях она выдает неправильный результат. Причина в том, что вычислительный механизм Excel не учитывает диапазоны, которые не передаются в качестве аргументов. Вследствие этого иногда перед возвратом функцией значения не вычисляются все связанные величины. Следует также написать функцию DoubleCell, в качестве аргумента которой передается значение ячейки A1.

```
Function DoubleCell(cell)
   DoubleCell = cell * 2
End Function
```

Функция с двумя аргументами

Представим, что менеджер, о котором речь шла выше, внедряет новую политику, разработанную для уменьшения текучести кадров: общая сумма комиссионных, подлежащих выплате, увеличивается на 1% за каждый год, который служащий проработал в компании.

Изменим пользовательскую функцию Commission (описанную в предыдущем разделе) так, чтобы она принимала два аргумента. Новый аргумент представляет количество лет, отработанных сотрудником в компании. Назовем эту новую функцию Commission2.

```
Function Commission2(Sales, Years)
'  Вычисляет комиссионные с продаж на основе
'  рабочего стажа в компании
   Const Tier1 = 0.08
   Const Tier2 = 0.105
   Const Tier3 = 0.12
   Const Tier4 = 0.14
   Select Case Sales
      Case 0 To 9999.99: Commission2 = Sales * Tier1
      Case 1000 To 19999.99: Commission2 = Sales * Tier2
      Case 20000 To 39999.99: Commission2 = Sales * Tier3
      Case Is >= 40000: Commission2 = Sales * Tier4
   End Select
   Commission2 = Commission2 + (Commission2 * Years / 100)
End Function
```

Довольно просто, правда? Добавим второй аргумент (Years) в оператор Function и применим дополнительные вычисления, изменяющие формулу комиссионных.

Далее приведен пример написания формулы с использованием этой функции (предполагается, что объем продаж задан в ячейке A1, а количество лет, которые проработал в компании рассматриваемый служащий, указано в ячейке B1).

```
=Commission2(A1,B1)
```

Компакт-диск

Все описанные выше процедуры вычисления комиссионных находятся на прилагаемом к книге компакт-диске, в файле `comission functions.xlsm`.

Функция с аргументом в виде массива

В качестве аргументов функции могут принимать один или несколько массивов, обрабатывать этот массив (массивы) и возвращать единственное значение.

Функция, представленная ниже, принимает как аргумент массив и возвращает сумму его элементов.

```
Function SumArray(List) As Double
   Dim Item As Variant
   SumArray = 0
   For Each Item In List
      If WorksheetFunction.IsNumber(Item) Then _
         SumArray = SumArray + Item
   Next Item
End Function
```

Функция Excel ЕЧИСЛО (`IsNumber`) проверяет, является ли каждый элемент числом, прежде чем добавить его к общему целому. Добавление этого простого оператора проверки данных устраняет ошибки несоответствия типов при попытке выполнить арифметическую операцию над строкой.

Следующая процедура демонстрирует, как вызвать эту функцию в процедуре. Процедура `MakeList` создает 100-элементный массив и присваивает каждому элементу массива случайное число. Функция `MsgBox` отображает сумму значений массива в результате вызова функции `SumArray`.

```
Sub MakeList()
   Dim Nums(1 To 100) As Double
   Dim i as Integer
   For i = 1 To 100
      Nums(i) = Rnd * 1000
   Next i
   MsgBox SumArray(Nums)
End Sub
```

Так как функция `SumArray` не объявляет тип данных своего аргумента (и аргумент имеет тип `Variant`), она работает и в формулах рабочего листа. Например, следующая формула возвращает сумму значений в диапазоне A1:C10.

```
=SumArray(A1:C10)
```

Возможно, вы заметили, что при использовании в формуле функция `SumArray` подобна функции Excel СУММ. Единственное отличие заключается в том, что функция `SumArray` принимает только один аргумент. Не стоит забывать, что этот пример приведен только в целях обучения. Функция `SumArray` в формуле не имеет абсолютно никаких преимуществ перед функцией Excel СУММ.

Компакт-диск

Этот пример под названием array argument.xlsm находится на прилагаемом к книге компакт-диске.

Функция с необязательными аргументами

Многие встроенные функции Excel имеют необязательные аргументы. Пример — функция ЛЕВСИМВ, возвращающая символы с левого края строки. Она имеет следующий синтаксис:

```
ЛЕВСИМВ(текст, кол_символов)
```

Первый аргумент — обязательный, в отличие от второго. Если не указан второй аргумент, Excel предполагает значение 1. Следовательно, две формулы, приведенные ниже, возвращают одинаковые результаты.

```
=ЛЕВСИМВ(A1,1)
=ЛЕВСИМВ(A1)
```

Пользовательские функции, разработанные в VBA, также могут иметь необязательные аргументы. Необязательный аргумент вы зададите, если введете перед именем аргумента ключевое слово `Optional`. В списке аргументов необязательные аргументы определяются после всех обязательных.

Далее приведен пример пользовательской функции с необязательным аргументом.

```
Function User(Optional UpperCase As Variant)
   If IsMissing(UpperCase) Then UpperCase = False
   User = Application.UserName
   If UpperCase Then User = UCase(User)
End Function
```

Если аргумент равен `False` или опущен, то имя пользователя возвращается без каких-либо изменений. Если же аргумент функции `True`, то имя пользователя возвращается в символах верхнего регистра (с помощью VBA-функции `Ucase`). Обратите внимание на первый оператор функции — он содержит VBA-функцию `IsMissing`, которая определяет наличие аргумента. Если аргумент отсутствует, оператор присваивает переменной `UpperCase` значение `False` (задано по умолчанию).

Приведенные ниже формулы справедливы, а первые две возвращают одинаковые результаты.

```
=User()
=User(False)
=User(True)
```

Примечание

Если нужно определить, подставляется ли необязательный аргумент в функцию, объявите его с типом данных `Variant`. Затем можно использовать встроенную функцию `IsMissing`, как показано в этом примере.

Далее приведен пример еще одной пользовательской функции с необязательным аргументом. Эта функция случайным образом выбирает одну ячейку из диапазона данных и возвращает содержимое этой ячейки. Если второй аргумент имеет значение `True`, то выделенное значение изменяется при каждом пересчете рабочего листа. Если второй аргумент имеет значение `False` (или не задан), функция не пересчитывается, кроме тех случаев, когда изменяется одна из ячеек диапазона введенных данных.

```
Function DrawOne(Rng As Variant, Optional Recalc As Variant = False)
'  Случайным образом выбирает одну ячейку из диапазона;
'  функция изменяемая, если аргумент Recalc имеет значение True
   Application.Volatile Recalc
'  Определение случайной ячейки
```

```
   DrawOne = Rng(Int((Rng.Count) * Rnd + 1))
End Function
```

Обратите внимание, что второй аргумент функции `Draw` содержит ключевое слово `Optional`, а также значение, заданное по умолчанию.

Все приведенные ниже формулы корректны, причем первые две возвращают одинаковые результаты.

```
=DrawOne(A1:A100)
=DrawOne(A1:A100,False)
=DrawOne(A1:A100,True)
```

Эта функция может быть полезной для выбора лотерейных номеров, победителя из списка имен и т.д.

Компакт-диск

Приведенная выше функция доступна на прилагаемом к книге компакт-диске и находится в файле `draw.xlsm`.

Функция VBA, возвращающая массив

VBA содержит весьма полезную функцию с названием `Array`. Она возвращает значение с типом данных `Variant`, которое содержит массив (т.е. несколько значений). Если вы знакомы с формулами массивов в Excel, то сможете легко разобраться в функции `Array` в VBA. Формула массива вводится в ячейку после нажатия <Ctrl+Shift+Enter>. Excel добавляет вокруг формулы скобки, чтобы указать, что это формула массива.

Перекрестная ссылка

Более подробную информацию о формулах массивов вы найдете в главе 3.

Примечание

Важно понимать, что массив, возвращаемый функцией `Array`, — это не тот обычный массив, который состоит из элементов с типом данных `Variant`. Другими словами, массив `Variant` существенно отличается от массива значений типа `Variant`.

Функция `MonthNames`, приведенная ниже, — простой пример применения функции `Array` в пользовательской функции.

```
Function MonthNames()
   MonthNames = Array("Янв", "Фев", "Мар", "Апр", _
      "Май", "Июн", "Июл", "Авг", "Сен", "Окт", _
      "Ноя", "Дек")
End Function
```

Функция `MonthNames` возвращает горизонтальный массив названий месяцев. Вы можете создать формулу массива для нескольких ячеек, используя функцию `MonthNames`. Прежде чем ее использовать, убедитесь, что в модуле VBA введен код функции. Затем на рабочем листе выделите несколько ячеек в строке (для начала выделите 12 ячеек), введите следующую формулу и нажмите <Ctrl+Shift+Enter>.

```
{=MonthNames()}
```

Если необходимо сгенерировать вертикальный массив названий месяцев, выделите вертикальный диапазон, введите следующую формулу и нажмите <Ctrl+Shift+Enter>.

```
{=ТРАНСП(MonthNames())}
```

Данная формула использует функцию Excel ТРАНСП для преобразования горизонтального массива в вертикальный.

Следующий пример — вариация на тему функции MonthNames.

```
Function MonthNames(Optional MIndex)
   Dim AllNames As Variant
   Dim MonthVal As Long
   AllNames = Array("Янв", "Фев", "Мар", "Апр", _
      "Май", "Июн", "Июл", "Авг", "Сен", "Окт", _
      "Ноя", "Дек")
   If IsMissing(MIndex) Then
      MonthNames = AllNames
   Else
      Select Case Mindex
         Case Is >= 1
         '  Определить значение месяца (например, 13=1)
         MonthVal = ((MIndex - 1) Mod 12)
         MonthNames = AllNames(MonthVal)
      Case Is <= 0 ' Вертикальный массив
         MonthNames = Application.Transpose(AllNames)
      End Select
   End If
End Function
```

Обратите внимание, что для проверки незаданного аргумента используется функция VBA IsMissing. В данной ситуации невозможно задать значение по умолчанию для незаданного аргумента в списке аргументов функции, так как значение по умолчанию определяется в функции. Функцию IsMissing можно использовать, только если необязательный аргумент имеет тип Variant.

Эта усовершенствованная функция использует необязательный аргумент, который необходим для выполнения некоторых задач.

- **Если аргумент не задан**, функция возвращает горизонтальный массив названий месяцев.
- **Если аргумент меньше или равен 0**, функция возвращает вертикальный массив названий месяцев. Для преобразования массива используется функция Excel ТРАНСП (Transpose).
- **Если аргумент больше или равен 1**, функция возвращает название месяца, соответствующее значению аргумента.

Примечание

В этой процедуре используется оператор Mod для определения значения месяца. Этот оператор возвращает остаток от деления первого операнда на второй. Учтите, что нумерация индексов в массиве AllNames начинается с нуля, а их диапазон простирается от 0 до 11. Поэтому из аргумента функции вычитается единица. Таким образом, аргумент 13 возвращает 0 (январь), аргумент 24 — 11 (декабрь).

Эту функцию можно использовать разными способами, как показано на рис. 10.6.

	A	B	C	D	E	F	G	H	I	J	K	L	M
1	Янв	Фев	Мар	Апр	Май	Июн	Июл	Авг	Сен	Окт	Ноя	Дек	
2													
3		1 Янв		Янв		Мар							
4		2 Фев		Фев									
5		3 Мар		Мар									
6		4 Апр		Апр									
7		5 Май		Май									
8		6 Июн		Июн									
9		7 Июл		Июл									
10		8 Авг		Авг									
11		9 Сен		Сен									
12		10 Окт		Окт									
13		11 Ноя		Ноя									
14		12 Дек		Дек									
15													

Рис. 10.6. Несколько способов передачи в рабочий лист массива или одного значения

Диапазон A1:L1 содержит следующую формулу, введенную как массив. Для начала выделите A1:L1, введите формулу и нажмите <Ctrl+Shift+Enter>.

```
{=MonthNames()}
```

В диапазоне A3:A14 находятся целые числа от 1 до 12. Ячейка B3 содержит обычную формулу, которая скопирована в 11 ячеек, следующих за ней.

```
=MonthNames(A3)
```

В диапазоне D3:D14 располагается формула, введенная как массив.

```
{=MonthNames(-1)}
```

Диапазон F3 содержит следующую формулу (не массив):

```
=MonthNames(3)
```

Примечание

Помните, что для ввода формулы массива нужно нажать клавиши <Ctrl+Shift+Enter> (не вводите фигурные скобки вручную).

Примечание

Нижняя граница массива, созданная с помощью функции `Array`, определяется нижней границей, заданной в операторе `Option Base` в верхней части модуля. Если оператор `Option Base` не задан, то по умолчанию нижняя граница равна 0.

Компакт-диск

Рабочая книга (`month names.xslm`), демонстрирующая функцию `MonthNames` в действии, находится на прилагаемом к книге компакт-диске.

Функция, возвращающая значение ошибки

В некоторых случаях необходимо, чтобы пользовательская функция возвращала значение ошибки. Рассмотрим функцию `RemoveVowels`, которая была представлена ранее.

```
Function RemoveVowels(Txt) As String
' Удаляет все гласные буквы из текстового аргумента
  Dim i As Long
  RemoveVowels = ""
  For i = 1 To Len(Txt)
    If Not UCase(Mid(Txt, i, 1)) Like "[AEIOUАЕИОУЫЭЮЯ]" Then
```

```
            RemoveVowels = RemoveVowels & Mid(Txt, i, 1)
        End If
    Next i
End Function
```

При использовании в составе формулы рабочего листа эта функция удаляет гласные буквы из аргумента, находящегося в одной ячейке. Если в качестве аргумента используется числовое значение, возвращается строка. В этом случае предпочтительнее возвращать сообщение об ошибке (#Н/Д).

Возможно, потребуется создать строку, которая выглядит как значение ошибки в формуле Excel.

```
RemoveVowels = "#Н/Д"
```

Несмотря на то что строка выглядит как значение ошибки, она не обрабатывается как ошибка другими формулами, которые могут на нее ссылаться. Чтобы получить в результате выполнения функции настоящее значение ошибки, используйте функцию `Cover`, которая преобразует номер ошибки в настоящую ошибку.

К счастью, в VBA содержатся встроенные константы для обозначения ошибок, которые должна возвращать пользовательская функция. Эти значения — ошибки выполнения формул Excel, а не ошибки выполнения кода VBA. Ниже перечислены встроенные константы ошибок:

- `xlErrDiv0` (для ошибки #ДЕЛ/0!);
- `xlErrNA` (для ошибки #Н/Д);
- `xlErrName` (для ошибки #ИМЯ?);
- `xlErrNull` (для ошибки #ПУСТО!);
- `xlErrNum` (для ошибки #ЧИСЛО!);
- `xlErrRef` (для ошибки #ССЫЛ!);
- `xlErrValue` (для ошибки #ЗНАЧ!).

Чтобы получить ошибку #Н/Д в пользовательской функции, примените следующий оператор:

```
RemoveVowels = CVErr(xlErrNA)
```

Ниже приведена преобразованная функция `RemoveVowels`. Конструкция `If-Then` применяется для выполнения альтернативного действия в случае, когда аргумент не является текстовым. Эта функция вызывает функцию Excel `ЕТЕКСТ (IsText)`, которая определяет, содержит ли аргумент текст. Если ячейка содержит текст, то функция возвращает нормальный результат. Если же ячейка содержит не текст (или пуста), то функция возвращает ошибку #Н/Д. Если ячейка не содержит текст (или пуста), возвращается ошибка #Н/Д.

```
Function RemoveVowels(Txt) As Variant
'  Удаляет все гласные буквы из текстового аргумента;
'  возвращает ошибку #ЗНАЧ!, если аргумент — не строка
   Dim i As Long
   RemoveVowels = ""
   If Application.WorksheetFunction.IsText(Txt) Then
      For i = 1 To Len(Txt)
         If Not UCase(Mid(Txt, i, 1)) Like "[AEIOUAЕИОУЫЭЮЯ]" then
            RemoveVowels = RemoveVowels & Mid(Txt, i, 1)
         End If
```

```
    Next i
  Else
    RemoveVowels = CVErr(xlErrNA)
  End If
End Function
```

Примечание

Обратите внимание, что был также изменен тип данных для возвращаемого функцией значения. Поскольку функция может возвращать что-то еще, кроме строки, тип данных был изменен на `Variant`.

Функция с неопределенным количеством аргументов

Некоторые функции Excel имеют неопределенное количество аргументов. Знакомый пример — функция СУММ, которая имеет следующий синтаксис:

```
СУММ(число1, число2...)
```

Первый аргумент обязателен. Кроме него, можно воспользоваться 254 дополнительными аргументами. Ниже приведен пример функции СУММ с четырьмя аргументами-диапазонами.

```
=СУММ(A1:A5,C1:C5,E1:E5,G1:G5)
```

Допускается использовать в функции разные типы аргументов. Например, следующий пример иллюстрирует использование трех аргументов: первый — диапазон, второй — значение, а третий — выражение.

```
=СУММ(A1:A5,12,24*3)
```

Существует возможность создавать функции, имеющие неопределенное количество аргументов. Основная идея заключается в следующем: примените в качестве последнего (или единственного) массив и добавьте перед ним ключевое слово `ParamArray`.

Примечание

`ParamArray` относится только к последнему аргументу в списке аргументов процедуры. Он всегда имеет тип данных `Variant` и всегда является необязательным аргументом (хотя ключевое слово `Optional` не используется).

Ниже представлена функция, которая может иметь произвольное количество одномерных аргументов (она не поддерживает многомерные аргументы-диапазоны). Функция возвращает сумму аргументов.

```
Function SimpleSum(ParamArray arglist() As Variant) As Double
  For Each arg In arglist
    SimpleSum = SimpleSum + arg
  Next arg
End Function
```

Для изменения этой функции в целях поддержки многомерных диапазонов следует добавить еще один цикл, который обрабатывает каждую ячейку в каждом из аргументов.

```
Function SimpleSum(ParamArray arglist() As Variant) As Double
  Dim cell As Range
  For Each arg In arglist
    For Each cell In arg
      SimpleSum = SimpleSum + cell
    Next cell
```

```
    Next arg
End Function
```

Функция SimpleSum далеко не такая гибкая, как функция Excel СУММ. Протестируйте ее на практике с разными типами аргументов, и вы убедитесь, что функция выдает ошибку, если все аргументы не являются либо числовым значением, либо ссылкой на ячейки, содержащие числовые значения.

Имитация функции СУММ

В данном разделе представлена пользовательская функция MySum. В отличие от функции SimpleSum, рассмотренной в предыдущем разделе, функция MySum идеально копирует поведение функции Excel СУММ.

Перед рассмотрением кода функции MySum уделим внимание функции Excel СУММ. Данная функция имеет очень широкие возможности. Она насчитывает до 255 аргументов (даже пропущенные аргументы), причем аргументами могут выступать числовые значения, ячейки, диапазоны, представленные в виде текста числа, логические значения и даже встроенные функции. Рассмотрим следующую формулу:

```
=СУММ(B1,5,"6",,ИСТИНА,КОРЕНЬ(4),A1:A5,D:D,C2*C3)
```

Эта формула содержит все типы аргументов, перечисленные в порядке их использования в функции:

- ссылка на одну ячейку;
- символьное значение;
- строка, которая выглядит как числовое значение;
- пропущенный аргумент;
- логическое значение ИСТИНА;
- выражение, использующее другую функцию;
- ссылка на диапазон.

Функция MySum (листинг 10.1) также обрабатывает все эти типы аргументов.

Компакт-диск

Рабочая книга, включающая функцию MySum, находится на прилагаемом к книге компакт-диске (файл mysum function.xlsm).

Листинг 10.1. Функция MySum

```
Function MySum(ParamArray args() As Variant) As Variant
'  Имитация функции Excel СУММ

'  Объявление переменных
   Dim i As Variant
   Dim TempRange As Range, cell As Range
   Dim ECode As String
   Dim m, n
   MySum = 0
'  Обработка каждого аргумента
   For i = 0 To UBound(args)
   '  Пропуск отсутствующих аргументов
      If Not IsMissing(args(i)) Then
```

```
      '  Каков тип аргумента?
         Select Case TypeName(args(i))
            Case "Range"
            '  Создание временного диапазона для строки/столбца
               Set TempRange = _
                  Intersect(args(i).Parent.UsedRange, _
               args(i))
               For Each cell In TempRange
                  If IsError(cell) Then
                     MySum = cell ' возвращается ошибка
                     Exit Function
                  End If
                  If cell = True Or cell = False Then
                     MySum = MySum + 0
                  Else
                     If IsNumeric(cell) Or IsDate(cell) _
                        then MySum = MySum + cell
                     End If
                  Next cell
               Case "Variant()"
                  n = args(i)
                  For m = LBound(n) To UBound(n)
                     MySum = MySum(MySum, n(m))
                     ' рекурсивный вызов
                  Next m
               Case "Null" ' игнорировать значение
               Case "Error" ' возвращение к ошибке
                  MySum = args(i)
                  Exit Function
               Case "Boolean"
               '  Проверка строкового значения
                  If args(i) = "True" Then MySum = MySum + 1
               Case "Date"
                  MySum = MySum + args(i)
               Case Else
                  MySum = MySum + args(i)
            End Select
      End If
   Next i
End Function
```

На рис. 10.7 показана рабочая книга с различными формулами, среди которых — функции СУММ и MySum. Как видите, эти функции возвращают одинаковые результаты.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
1		1	ИСТИНА	Первый	1	Один	5:00 PM			1						
2		4	ЛОЖЬ	"2"	#Н/Д	Два	04.янв									
3		2	ИСТИНА	3	3	Три	05.янв									
4																
5	СУММ -->	7	0	3	#Н/Д	0	1.11.18 5:00 PM	1	1	100	11,48913	31	#ДЕЛ/0!	#ЗНАЧ!	3,708333	
6																
7	MYSUM -->	7	0	3	#Н/Д	0	1.11.18 5:00 PM	1	1	100	11,48913	31	#ДЕЛ/0!	#ЗНАЧ!	3,708333	
8																

Рис. 10.7. Сравнение функций СУММ и MySum

Если вы хотите посмотреть, как работает эта функция, создайте использующую ее формулу. Затем добавьте в код точку останова и приступите к построчному выполнению операторов (см. раздел "Отладка функций"). Протестируйте различные типы аргументов,

что позволит гарантировать работоспособность функции. В ходе анализа кода функции MySum помните о следующем.

- Пропущенные аргументы (определенные с помощью функции IsMissing) игнорируются.
- Процедура использует для определения типа аргумента (Range, Error и т.д.) функцию VBA TypeName. Каждый тип аргумента обрабатывается определенным способом.
- Для аргумента-диапазона функция циклически просматривает все ячейки диапазона и прибавляет их значения к сумме.
- Функция имеет тип данных Variant, так как она должна возвращать ошибку, если один из ее аргументов имеет значение ошибки.
- Если аргумент содержит ошибку (например, #ДЕЛ!/0), то функция MySum возвращает ошибку — как и функция Excel СУММ.
- Функция Excel СУММ "полагает", что текстовая строка имеет значение 0, если только она не является символьным аргументом (т.е. фактическим значением, а не переменной). Следовательно, функция MySum прибавляет значение ячейки лишь в том случае, если его можно оценить как число (для этого применяется функция VBA IsNumeric).
- Для аргументов-диапазонов функция использует метод Intersect с целью создания временного диапазона, который состоит из пересечения заданного диапазона и используемого диапазона листа. В результате обрабатываются те случаи, когда аргумент состоит из полной строки или столбца, а его просмотр будет длиться вечность.

Возможно, вас интересует относительная скорость выполнения функций СУММ и MySum. Конечно, функция MySum значительно медленнее, ее скорость зависит от быстродействия вашей системы и самих формул. Если в системе рабочий лист, содержащий пять тысяч формул СУММ, вычисляется за долю секунды, то после замены функций СУММ на функции MySum пересчет занимает 8 секунд.

Функция MySum несколько улучшена, но ее производительность все же намного меньше производительности функции СУММ. Цель этого примера состоит не в том, чтобы создать новую функцию СУММ. Скорее, этот пример показывает, как создавать пользовательские функции рабочих листов, которые выглядят и работают так же, как встроенные функции Excel.

Расширенные функции для работы с датами

Достаточно часто пользователи Excel жалуются на невозможность работы с датами до 1900 года. Например, к таким пользователям относятся специалисты по созданию генеалогических деревьев, которые часто применяют Excel для отслеживания дат рождения и смерти. Если какая-либо из подобных дат попадает в период до 1900 года, вычисление количества прожитых лет становится невозможным.

Я создал набор функций, использующих возможности VBA по работе с широчайшим диапазоном дат. Наиболее ранняя дата, распознаваемая VBA, — 1 января 100 года.

Предупреждение

Учитывайте изменения календарей. Осторожно работайте с датами до 1752 года. Различия в исторических американских, британских, григорианских и юлианских календарях могут привести к некорректным результатам вычислений.

Ниже приведены функции VBA для работы с датами.

- **`XDATE(y,m,d,fmt)`.** Возвращает дату, "сконструированную" на основе введенных пользователем года, месяца и дня. Дополнительно можно указать строку форматирования даты.
- **`XDATEADD(xdate1,days,fmt)`.** Добавляет к дате указанное количество дней. Дополнительно можно указать строку форматирования даты.
- **`XDATEDIF(xdate1,xdate2)`.** Возвращает количество дней между двумя датами.
- **`XDATEYEARDIF(xdate1,xdate2)`.** Возвращает количество полных лет между двумя датами (применяется для вычисления возраста).
- **`XDATEYEAR(xdate1)`.** Возвращает год даты.
- **`XDATEMONTH(xdate1)`.** Возвращает месяц даты.
- **`XDATEDAY(xdate1)`.** Возвращает день даты.
- **`XDATEDOW(xdate1)`.** Возвращает день недели даты (в виде целого числа между 1 и 7).

На рис. 10.8 показана рабочая книга, использующая некоторые из описанных выше функций.

	A	B	C	D	E	F	G	H	I
5									
6	Президент	Год	Месяц	День	XDATE	XDATEDIF	XDATEYEARDIF	XDATEDOW	
7	Джордж Вашингтон	1732	2	22	Февраль 22, 1732	101 706	278	Пятница	
8	Джон Адамс	1735	10	30	Октябрь 30, 1735	100 360	274	Воскресенье	
9	Томас Джефферсон	1743	4	13	Апрель 13, 1743	97 638	267	Суббота	
10	Джеймс Мэдисон	1751	3	16	Март 16, 1751	94 744	259	Вторник	
11	Джеймс Монро	1758	4	28	Апрель 28, 1758	92 144	252	Пятница	
12	Джон Куинси Адамс	1767	7	11	Июль 11, 1767	88 783	243	Суббота	
13	Эндрю Джексон	1767	3	15	Март 15, 1767	88 901	243	Воскресенье	
14	Мартин Ван Бюрен	1782	12	5	Декабрь 5, 1782	83 157	227	Четверг	
15	Уильям Генри Гаррисон	1773	2	9	Февраль 9, 1773	86 743	237	Вторник	
16	Джон Тайлер	1790	3	29	Март 29, 1790	80 486	220	Понедельник	
17	Джеймс Нолк Полк	1795	11	2	Ноябрь 2, 1795	78 442	214	Понедельник	
18	Закари Тейлор	1784	11	24	Ноябрь 24, 1784	82 437	225	Среда	
19	Миллорд Филлмор	1800	1	7	Январь 7, 1800	76 915	210	Вторник	
20	Франклин Пирс	1804	11	23	Ноябрь 23, 1804	75 134	205	Пятница	
21	Джеймс Бьюкенен	1791	4	23	Апрель 23, 1791	80 096	219	Суббота	
22	Авраам Линкольн	1809	2	12	Февраль 12, 1809	73 592	201	Воскресенье	
23	Эндрю Джонсон	1808	12	29	Декабрь 29, 1808	73 637	201	Четверг	
24	Улисс С. Грант	1822	4	27	Апрель 27, 1822	68 770	188	Суббота	
25	Раттерфорд Б. Хейс	1822	10	4	Октябрь 4, 1822	68 610	187	Пятница	
26	Джеймс А. Гарфилд	1831	11	19	Ноябрь 19, 1831	65 277	178	Суббота	
27	Честер А. Артур	1829	10	5	Октябрь 5, 1829	66 052	180	Понедельник	

Рис. 10.8. Функции VBA, предназначенные для работы с датами

Учтите, что возвращаемая функциями дата представляет собой текстовую строку, а не реальную дату. Поэтому с данным значением невозможно выполнять математические операции с помощью стандартных операторов Excel. Но зато это значение можно использовать в качестве аргумента для других функций VBA, предназначенных для работы с датами.

Описанные в этом разделе функции весьма просты. Для примера ниже приводится листинг функции XDATE.

```
Function XDATE(y, m, d, Optional fmt As String) As String
   If IsMissing(fmt) Then fmt = "Short Date"
   XDATE = Format(DateSerial(y, m, d), fmt)
End Function
```

Ниже описываются аргументы функции XDATE:

- `y` — состоящий из четырех цифр год, находящийся в диапазоне от 0100 до 9999;
- `m` — номер месяца (1–12);
- `d` — номер дня (1–31);
- `fmt` (необязательно) — строка форматирования даты.

Если аргумент `fmt` не указан, отображение даты осуществляется в формате короткой даты, установленном в системе (на панели управления Windows).

Если аргумент `m` или `d` превышает допустимое значение, происходит переход к следующему году или месяцу. Например, если указать месяц с номером 13, он будет интерпретироваться как январь будущего года.

Компакт-диск

Код VBA для расширенных функций по работе с датами находится на прилагаемом к книге компакт-диске в файле `extended date function.xlsm`. Здесь же можно найти инструкции по работе с этими функциями — в файле `extended date functions help.docx`.

Отладка функций

При использовании формулы на рабочем листе для тестирования функции происходящие в процессе выполнения ошибки не отображаются в знакомом диалоговом окне сообщений. Формула просто возвращает значение ошибки (`#ЗНАЧ!`). К счастью, это не представляет большой проблемы при отладке функций, так как всегда существует несколько "обходных путей".

- **Поместить в стратегически важных местах функцию `MsgBox`, чтобы контролировать значения отдельных переменных.** Удобно, если в процессе выполнения функций окна сообщений все-таки появляются, в отличие от окон ошибок. Убедитесь, что ваша функция используется только в одной формуле на рабочем листе, иначе окна сообщений будут появляться для каждой такой формулы (подобное поведение быстро надоедает).
- **Протестировать функцию, вызвав ее из процедуры, а не в формуле рабочего листа.** Ошибки в процессе выполнения отображаются обычным образом, поэтому можно либо сразу решить проблему (если она вам известна), либо перейти к отладке.
- **Определить точку остановки в функции и просмотреть функцию пошагово.** При этом можно воспользоваться всеми стандартными инструментами отладки. Чтобы добавить точку остановки, поместите курсор в операторе, в котором вы решили приостановить выполнение, и выберите команду Debug⇨Toggle Breakpoint (Отладка⇨Точка остановки) (или нажмите <F9>).

- **Использовать в программе один или несколько временных операторов Debug⇨Print (Отладка⇨Печать), чтобы отобразить значения в окне Immediate редактора VBA.** Например, чтобы проконтролировать циклически изменяемое значение, используйте следующий метод.

```
Function VowelCount(r) As Long
   Dim Count As Long
   Dim i As Long
   Dim Ch As String * 1
   Count = 0
   For i = 1 To Len(r)
      Ch = UCase(Mid(r, i, 1))
      If Ch Like "[AEIOUАЕИОУЫЭЮЯ]" Then
         Count = Count + 1
         Debug.Print Ch, i
      End If
   Next i
   VowelCount = Count
End Function
```

В данном случае значения двух переменных, `Ch` и `i`, выводятся в окне отладки (Immediate) всякий раз, когда в программе встречается оператор Debug⇨Print. На рис. 10.9 показан результат для случая, когда функция принимает аргумент `TucsonArizona`.

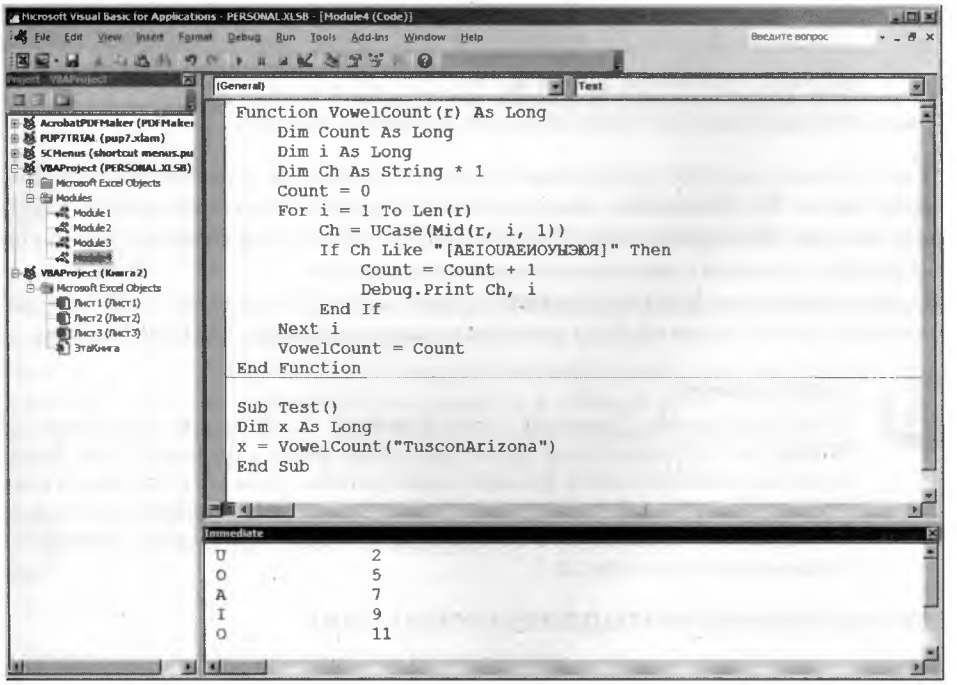


Рис. 10.9. Используйте окно отладки для отображения результатов при выполнении функции

Работа с диалоговым окном Мастер функций

Диалоговое окно Excel Мастер функций (Insert Function) — очень удобный инструмент. При создании формулы рабочего листа он позволяет выбрать необходимую функцию из списка. Функции в списке группируются в разные категории, благодаря чему об-

легчается их поиск. После выбора функции и щелчка на кнопке OK появляется диалоговое окно Аргументы функции (Function Arguments), с помощью которого облегчается вставка аргументов функции. На рис. 10.10 показаны оба этих диалоговых окна.

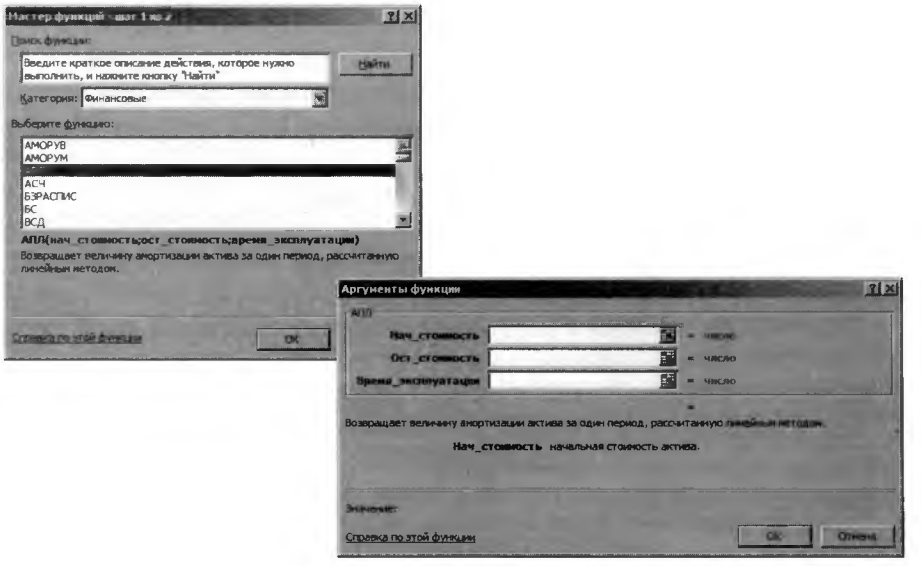


Рис. 10.10. Диалоговое окно Мастер функций облегчает вставку функции в формулу, а окно Аргументы функции позволяет задать аргументы функции

В диалоговом окне Мастер функций также отображаются пользовательские функции рабочего листа. По умолчанию пользовательские функции отображаются в категории Определенные пользователем (User Defined). В диалоговом окне Аргументы функции (Function Arguments) указываются аргументы функции.

В диалоговом окне Мастер функций можно найти функцию по ключевому слову. К сожалению, найти таким образом пользовательскую функцию VBA невозможно.

Примечание

Пользовательские функции, определенные с помощью ключевого слова Private, не отображаются в диалоговом окне Мастер функций (Insert Function). Если вы разрабатываете функции исключительно для использования в других процедурах VBA, объявляйте их с помощью ключевого слова `Private`. Такая функция не отображается в окне Мастер функций, хотя и может применяться в формулах рабочего листа.

Использование метода `MacroOptions`

Можно воспользоваться методом `MacroOptions` объекта `Application`, который позволяет включить в состав встроенных функций Excel разработанные вами функции. Этот метод позволяет выполнить следующее:

- добавить описание функции;
- указать категорию функции;
- добавить описание аргументов функции.

Ниже приводится пример процедуры, которая использует метод MacroOptions для указания необходимых сведений о функции.

```
Sub DescribeFunction()
   Dim FuncName As String
   Dim FuncDesc As String
   Dim FuncCat As Long
   Dim Arg1Desc As String, Arg2Desc As String

   FuncName = "Draw"
   FuncDesc = "Содержимое случайной ячейки диапазона"
   FuncCat = 5 'Ссылки и массивы
   Arg1Desc = "Диапазон, который содержит значения"
   Arg2Desc = "(Не обязательно) Если False или отсутствует, _
              новой ячейки нет"
   Arg2Desc = Arg2Desc & "выбрано при пересчете. Если True, "
   Arg2Desc = Arg2Desc & "выбрана новая ячейка при пересчете."

   Application.MacroOptions _
      Macro:=FuncName, _
      Description:=FuncDesc, _
      Category:=FuncCat, _
      ArgumentDescriptions:=Array(Arg1Desc, Arg2Desc) End Sub
```

Эта процедура использует переменные для хранения различной информации и в качестве аргументов метода MacroOptions. Функция относится к категории 5 (ссылки и массивы). Обратите внимание, что описания для двух аргументов указываются путем применения массива в качестве последнего аргумента метода MacroOptions.

Новинка

Поддержка описаний аргументов появилась в Excel 2010. Если рабочая книга открывается в более ранней версии Excel, описание аргументов не отображается.

На рис. 10.11 показаны диалоговые окна Мастер функций и Аргументы функции после вызова указанной выше процедуры.

Процедуру DescribeFunction следует вызывать только один раз. После ее вызова информация, связанная с функцией, сохраняется в рабочей книге. Можно также не указывать аргументы. Например, если для создания описаний аргументы не требуются, можно не обращать внимания на аргумент ArgumentDescriptions.

Дополнительные сведения о создании справки, доступной в окне Мастер функций, можно найти в главе 24.

Определение категории функции

Если вы не укажете категорию функции с помощью метода MacroOptions, пользовательская функция рабочего листа появится в категории Определенные пользователем (User Defined) диалогового окна Мастер функций (Insert Function). Иногда все же требуется поместить ее в другую категорию. Перемещение функции в другую категорию приведет к появлению ее в соответствующем раскрывающемся списке группы Формулы⇨Библиотека функций (Formulas⇨Function Library).

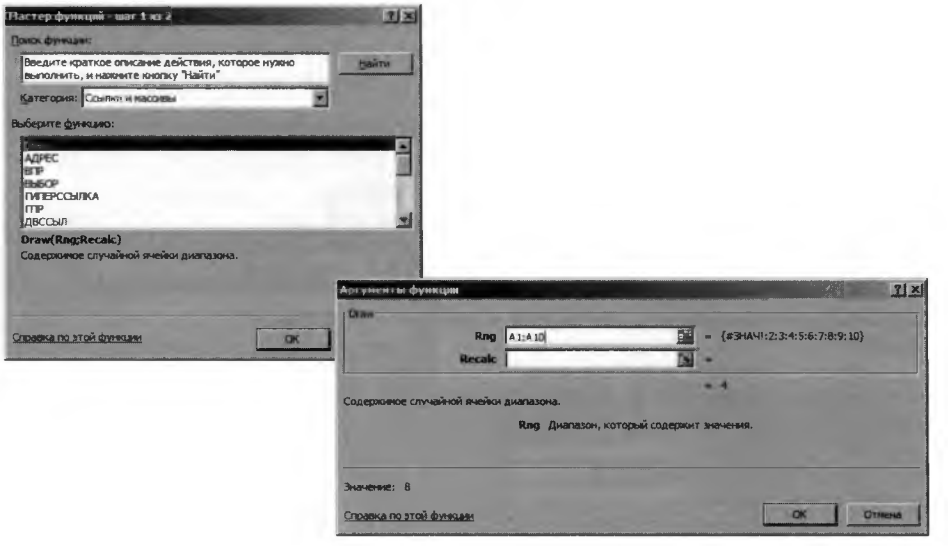


Рис. 10.11. Вид диалоговых окон Мастер функций и Аргументы функции для пользовательской функции

В табл. 10.1 перечислены номера категорий, которые можно использовать в качестве значений аргумента `Category` метода `MacroOptions`. Обратите внимание, что некоторые из этих категорий (от 10 до 13) обычно не отображаются в диалоговом окне Мастер функций (Insert Function). Если же отнести одну из пользовательских функций в подобную категорию, она появится в диалоговом окне.

Таблица 10.1. Категории функций

Номер категории	Название категории
0	Полный алфавитный перечень (All)
1	Финансовые (Financial)
2	Дата и время (Date & Time)
3	Математические (Math & Trig)
4	Статистические (Statistical)
5	Ссылки и массивы (Lookup & Reference)
6	Работа с базой данных (Database)
7	Текстовые (Text)
8	Логические (Logical)
9	Проверка свойств и значений (Information)
12	Управление макросами (Macro Control)
13	Динамический обмен данными/Внешние (DDE/External)
14	Определенные пользователем (User Defined)
15	Инженерные (Enhineering)
16	Аналитические (Cube)
17	Совместимость (Compatibility)*

*Категория Совместимость появилась в версии Excel 2010.

Можно самостоятельно создавать категории функций. Вместо указания номера в качестве значения аргумента `Category` метода `MacroOptions` воспользуйтесь текстовой строкой. Приведенный ниже оператор создает категорию функций **Функции VBA** и назначает эту категорию функции `Commission`.

```
Application.MacroOptions Macro:="Commission", Category:="Функции VBA"
```

Добавление описания функции вручную

В качестве альтернативы методу `MacroOptions` для добавления описания функции можно воспользоваться диалоговым окном **Макрос** (Macro).

Примечание

Если не указать описание пользовательской функции, в диалоговом окне **Мастер функций** (Insert Function) отобразится следующий текст: `Справка недоступна` (No help available).

Для добавления описания пользовательской функции выполните следующие действия.

1. Создайте функцию в VBE.
2. Активизируйте Excel, убедитесь в том, что требуемая функция находится в активной рабочей книге.
3. Выберите команду **Разработчик⇨Код⇨Макросы** (Developer⇨Code⇨Macros) или нажмите клавиши <Alt+F8>.

   В диалоговом окне **Макрос** перечислены доступные процедуры, но в этом списке вы можете не найти своих функций.
4. Введите название функции в поле **Название макроса** (Macro Name).
5. Щелкните на кнопке **Параметры** (Options) для отображения диалогового окна **Параметры макроса** (Macro Options).
6. Введите описание функции в поле **Описание** (Description) (рис. 10.9). Поле **Сочетание клавиш** (Shortcut Key) к описанию функций не относится.
7. Щелкните на кнопке **OK** и на кнопке **Отмена** (Cancel).

После выполнения описанных выше действий в диалоговом окне **Мастер функций** (Insert Function) отображается описание, которое было введено в п. 6 (при выборе функции).

Если для ввода функции используется диалоговое окно **Мастер функций**, то диалоговое окно **Аргументы функций** отображается после щелчка на кнопке **OK**. Для встроенных функций диалоговое окно **Аргументы функций** предоставляет описание каждого аргумента. К сожалению, отобразить такое описание для аргументов пользовательских функций невозможно.

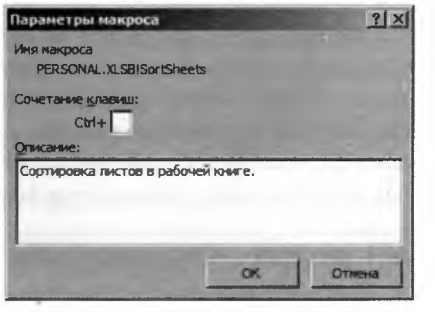


Рис. 10.12. Ввод описания функции

Использование надстроек для хранения пользовательских функций

При желании можно сохранить часто используемые пользовательские функции в файле надстройки. Основное преимущество такого подхода заключается в следующем: функции могут быть применены в формулах без спецификатора имени файла.

Предположим, у вас есть пользовательская функция `ZapSpaces`; она хранится в файле `Myfuncs.xlsm`. Чтобы применить ее в формуле другой рабочей книги (отличной от `Myfuncs.xlsm`), необходимо ввести следующую формулу:

```
=Myfuncs.xlsm!ZapSpaces(A1:C12)
```

Если вы создадите надстройку на основе файла `Myfuncs.xlsm` и эта надстройка будет загружена в текущем сеансе работы Excel, то ссылку на файл можно пропустить, введя следующую формулу:

```
=ZapSpaces(A1:C12)
```

Перекрестная ссылка

Подробнее надстройки будут рассмотрены в главе 21.

Предупреждение

Потенциальная проблема, которая может возникнуть из-за использования надстроек для хранения пользовательских функций, связана с зависимостью рабочей книги от файла надстроек. Если вы передаете рабочую книгу сотруднику, не забудьте также передать копию надстройки, которая содержит требуемые функции.

Использование функций Windows API

VBA может заимствовать методы из других файлов, которые не имеют ничего общего с Excel или VBA, например файлы DLL (Dynamic Link Library — динамически подключаемая библиотека), которые используются Windows и другими программами. В результате в VBA появляется возможность выполнять операции, которые без заимствованных методов находятся за пределами возможностей языка.

Windows API (Application Programming Interface — интерфейс прикладного программирования) представляет собой набор функций, доступных программистам в среде Windows. При вызове функции Windows из VBA вы обращаетесь к Windows API. Многие ресурсы Windows, используемые программистами Windows, можно получить из файлов *DLL*, в которых хранятся программы и функции, подсоединяемые в процессе выполнения программы, а не во время компиляции.

API-функции и 64-разрядная Excel

Разработчики кода VBA должны учитывать особенности 64-разрядной версии Excel 2010. Если нужно обеспечить совместимость кода с 32-разрядными версиями (включая Excel 2007) и 64-разрядной версией Excel 2010, API-функции следует объявлять дважды (используйте директивы компилятора, которые гарантируют корректность используемого объявления).

Например, приведенное ниже объявление совместимо с 32-разрядными версиями Excel, но выдает ошибку компиляции при использовании в 64-разрядной версии Excel 2010.

```
Declare Function GetWindowsDirectoryA Lib "kernel32" _
   (ByVal lpBuffer As String, ByVal nSize As Long) As Long
```

Во многих случаях для обеспечения совместимости объявления с 64-разрядной версией Excel достаточно воспользоваться ключевым словом `PtrSafe` вместо ключевого слова `Declare`. Приведенное ниже объявление совместимо как с 32-разрядной версией Excel 2010, так и с 64-разрядной версией Excel 2010.

```
Declare PtrSafe Function GetWindowsDirectoryA Lib "kernel32" _
   (ByVal lpBuffer As String, ByVal nSize As Long) As Long
```

Но при выполнении этого кода в Excel 2007 (и более ранних версиях) появится сообщение об ошибке, поскольку ключевое слово `PtrSafe` в этих версиях не распознается.

В главе 26 будут описаны методы обеспечения совместимости объявлений API-функций со всеми 32-разрядными версиями Excel и 64-разрядной версией Excel 2010.

Примеры функций Windows API

Прежде чем использовать функцию Windows API, ее необходимо объявить вверху программного модуля. Если программный модуль — это не стандартный модуль VBA (т.е. модуль для `UserForm`, `Лист` или `ЭтаКнига`), то API-функцию необходимо объявить как `Private`.

Объявление API-функции имеет некоторую сложность — функция должна объявляться максимально точно. Оператор объявления указывает VBA следующее:

- какую API-функцию вы используете;
- в какой библиотеке расположена API-функция;
- аргументы API-функции.

После объявления API-функцию можно использовать в программе VBA.

Определение папки Windows

В этом разделе представлен пример API-функции, которая отображает имя папки Windows (с помощью стандартных операторов VBA эту задачу порой выполнить невозможно). Этот код будет неработоспособным в Excel 2007.

Ниже показан пример объявления API-функции.

```
Declare Function GetWindowsDirectoryA Lib "kernel32" _
   (ByVal lpBuffer As String, ByVal nSize As Long) As Long
```

Эта функция, имеющая два аргумента, возвращает название папки, в которой установлена операционная система Windows (с помощью одной лишь программы VBA такое действие выполнить невозможно). После вызова этой функции путь к папке Windows будет храниться в переменной `lpBuffer`, а длина строки пути — в переменной `nSize`.

После вставки оператора `Declare` вверху модуля вы можете обратиться к функции `GetWindowsDirectoryA`. Ниже следует пример вызова функции и отображения результата в окне сообщения.

```
Sub ShowWindowsDir()
   Dim WinPath As String * 255
   Dim WinDir As String
```

```
    WinPath = Space(255)
    WinDir = Left(WinPath, GetWindowsDirectoryA _
       (WinPath, Len(WinPath)))
    MsgBox WinDir, vbInformation, "Папка Windows"
End Sub
```

В процессе выполнения процедуры `ShowWindowsDir` отображается окно сообщения с указанием расположения папки Windows.

Иногда требуется создать *оболочку* (wrapper) для API-функций. Другими словами, вы создадите собственную функцию, использующую API-функцию. Такой подход существенно упрощает использование API-функции. Ниже приведен пример такой функции VBA.

```
Function WindowsDir() As String
'   Возвращает путь к папке Windows
    Dim WinPath As String * 255
    WinPath = Space(255)
    WindowsDir = Left(WinPath, GetWindowsDirectoryA _
       (WinPath, Len(WinPath)))
End Function
```

После объявления этой функции можно вызвать ее из другой процедуры.

```
MsgBox WindowsDir()
```

Можно также использовать эту функцию в формуле рабочего листа.

```
=WindowsDir()
```

Компакт-диск

Пример файла `windows directory.xlsm` можно найти на прилагаемом к книге компакт-диске.

API-функции выполняют действия, которые другим способом выполнить невозможно (или очень сложно). Если приложение должно найти путь к папке Windows, вы можете целый день искать и не найти в Excel или VBA функцию, которая выполняет эту задачу. Но зная, как получить доступ к функциям Windows API, вы без особого труда решите свою проблему.

Предупреждение

Не удивляйтесь сбоям в системе при использовании в VBA функций Windows API. Заранее сохраните свою работу перед тестированием.

Определение состояния клавиши <Shift>

Приведем еще один пример. Предположим, вы написали макрос VBA, который будет выполняться с помощью кнопки на панели инструментов. Необходимо, чтобы этот макрос выполнялся по-другому, если пользователь после щелчка на кнопке удерживает клавишу <Shift>. Чтобы узнать о нажатии клавиши <Shift>, можно использовать API-функцию `GetKeyState`. Функция `GetKeyState` сообщает о том, нажата ли конкретная клавиша. Функция имеет один аргумент, `nVirtKey`, который представляет код интересующей вас клавиши.

Ниже приведена программа, которая выявляет, что при выполнении процедуры обработки события `Button_Click` была нажата клавиша <Shift>. Обратите внимание, что для определения состояния клавиши <Shift> используется константа (принимающая ше-

стнадцатеричное значение), которая затем применяется как аргумент функции GetKeyState. Если GetKeyState возвращает значение меньше 0, это означает, что клавиша <Shift> нажата; в противном случае клавиша <Shift> не нажата.

```
Declare Function GetKeyState Lib "user32" _
   (ByVal nVirtKey As Long) As Integer
Sub Button_Click()
   Const VK_SHIFT As Integer = &H10
   If GetKeyState(VK_SHIFT) < 0 Then
      MsgBox "Клавиша <Shift> нажата"
   Else
      MsgBox "Клавиша <Shift> не нажата"
   End If
End Sub
```

Компакт-диск

В рабочей книге key press.xlsm, которая находится на прилагаемом к книге компакт-диске, вы найдете примеры определения состояния следующих клавиш (и их комбинаций): <Ctrl>, <Shift> и <Alt>.

Дополнительная информация о функциях Windows API

Работа с функциями Windows API может быть довольно сложной. Во многих книгах по программированию перечислены операторы объявления API-функций с соответствующими примерами. Как правило, можно просто скопировать выражения объявления и использовать функции, не вникая в их суть. Большинство VBA-программистов в Excel рассматривают API-функции как панацею для решения большинства задач. В Интернете вы найдете сотни вполне надежных примеров, которые можно скопировать и вставить в собственную программу.

Перекрестная ссылка

В главе 11 можно найти ряд дополнительных примеров использования функций Windows API.

Компакт-диск

На прилагаемом к книге компакт-диске находится файл win32api.txt, представляющий собой текстовый файл, содержащий объявления и константы Windows API. Можно открыть этот файл в текстовом редакторе и скопировать соответствующие объявления в модуль VBA.

Глава 11

Приемы и методы программирования на VBA

В этой главе...

- Учимся на примерах
- Работа с диапазонами
- Управление рабочими книгами и листами
- Методы программирования на VBA
- Полезные функции для программ VBA
- Полезные функции в формулах Excel
- Вызов функций Windows API

В этой главе рассматриваются примеры, призванные закрепить теоретические знания по программированию на VBA.

Учимся на примерах

Изучение любого предмета, в том числе и программирования, можно сделать более эффективным, рассматривая примеры. Подобный подход используется и программистами на VBA. Хорошо продуманный пример, как правило, передает определенную идею лучше, чем лежащая в его основе теория. Поэтому я не ставил своей целью создать справочник, в котором скрупулезно описаны все нюансы работы с VBA. Вместо этого я подготовил ряд примеров, демонстрирующих полезные приемы программирования в Excel:

- примеры использования VBA для работы с диапазонами;
- примеры использования VBA для управления рабочими книгами и листами;
- пользовательские функции, используемые в процедурах VBA и формулах рабочего листа;

- методы и способы написания программ на VBA;
- примеры использования функций Windows API.

В предыдущих главах этой части был представлен вводный теоретический материал. Кроме того, в электронной справочной системе можно найти все детальные сведения, которые в книге опущены. В этой главе приведены примеры решения практических задач, благодаря которым вы сможете углубить свои познания в области VBA.

Примеры, приведенные в данной главе, можно разделить на шесть категорий:

- работа с диапазонами;
- управление рабочими книгами и листами;
- методы программирования на VBA;
- функции, используемые в процедурах VBA;
- функции, применяемые в формулах рабочего листа;
- вызов функций Windows API.

Перекрестная ссылка

В последующих главах будут представлены частные примеры управления диаграммами, сводными таблицами, формами, событиями и т.п.

Работа с диапазонами

Примеры данного раздела демонстрируют принципы управления диапазонами на рабочих листах с помощью VBA.

Копирование диапазона

Функция записи макросов Excel используется не столько для создания хорошего кода, сколько для поиска названий необходимых объектов, методов и свойств. Программа, полученная в результате записи макросов, не всегда самая эффективная, но обычно предоставляет немало полезных сведений.

Например, при записи простой операции копирования и вставки мною получено пять строк VBA-кода.

```
Sub Macro1()
    Range("A1").Select
    Selection.Copy
    Range("B1").Select
    ActiveSheet.Paste
    Application.CutCopyMode = False
End Sub
```

Использование примеров данной главы

Не все примеры в настоящей главе представляют независимые программы. Однако они являются выполняемыми процедурами, которые можно применять в собственных приложениях.

Настоятельно рекомендуется в процессе чтения данной главы работать за компьютером. Можете изменять примеры и наблюдать, что происходит. Этот практический опыт гарантированно даст вам больше, чем чтение справочников.

Обратите внимание, что данная программа выделяет ячейки. Однако в VBA для работы с объектом не обязательно его выделять. Вы никогда не узнали бы об этом важном моменте, если бы копировали показанный выше код макроса, где в двух строках содержится метод `Select`. Данную процедуру можно заменить значительно более простой — применить метод `Copy`, который использует аргумент, представляющий адрес места вставки копируемого диапазона.

```
Sub CopyRange()
   Range("A1").Copy Range("B1")
End Sub
```

В обоих макросах предполагается, что рабочий лист является активным и операция выполняется на активном рабочем листе. Чтобы скопировать диапазон на другой рабочий лист или в другую книгу, необходимо задать ссылку. В следующем примере диапазон из листа `Лист1` книги `File.xlsx` копируется на лист `Лист2` книги `File2.xlsx`. Так как ссылки заданы правильно, пример работает независимо от того, какая рабочая книга активна.

```
Sub CopyRange2()
   Workbooks("File1.xlsx").Sheets("Лист1").Range("A1").Copy _
      Workbooks("File2.xlsx").Sheets("Лист2").Range("A1")
End Sub
```

Еще одним подходом к решению этой задачи является использование для представления диапазонов объектных переменных, как показано в следующем примере.

```
Sub CopyRange3()
   Dim Rng1 As Range, Rng2 As Range
   Set Rng1 = Workbooks("File1.xlsx").Sheets("Лист1").Range("A1")
   Set Rng2 = Workbooks("File2.xlsx").Sheets("Лист2").Range("A1")
   Rng1.Copy Rng2
End Sub
```

Понятно, что копирование не ограничивается единственной ячейкой за операцию. Например, следующая процедура копирует большой диапазон. Помните, что адрес места вставки определяется единственной ячейкой (представляющей верхний левый угол вставляемого диапазона).

```
Sub CopyRange4()
   Range("A1:C800").Copy Range("D1")
End Sub
```

Перемещение диапазона

Инструкции VBA для перемещения диапазона ячеек подобны инструкциям копирования диапазона (см. следующий пример). Разница заключается в том, что вместо метода `Copy` используется метод `Cut`. Обратите внимание, что для вставляемого диапазона необходимо задавать только адрес левой верхней ячейки.

Следующий пример демонстрирует операцию перемещения 18 ячеек (в диапазоне A1:C6) на новое место, начиная с ячейки H1.

```
Sub MoveRange1()
   Range("A1:C6").Cut Range("H1")
End Sub
```

Копирование диапазона переменного размера

Во многих случаях необходимо скопировать диапазон ячеек, когда неизвестны точные размеры диапазона (количество столбцов и строк). Например, вы управляете рабочей книгой, в которой фиксируется объем продаж за неделю. Количество строк еженедельно меняется по мере добавления новых данных.

На рис. 11.1 показан стандартный вид рабочего листа. Диапазон состоит из нескольких строк, количество строк изменяется каждую неделю. А поскольку невозможно знать точный размер диапазона в определенный момент времени, процесс написания макроса, копирующего данный диапазон, несколько усложняется.

	A	B	C	D	E
1	**Неделя**	**Объем продаж**	**Новые заказчики**		
2	1	21093	45		
3	2	25375	49		
4	3	26180	38		
5	4	25564	32		
6	5	29325	22		
7	6	23069	23		
8	7	24281	53		
9					
10					
11					
12					

Рис. 11.1. Количество строк в диапазоне дат изменяется каждую неделю

Следующий макрос демонстрирует, как скопировать данный диапазон из листа `Лист1` на лист `Лист2` (начиная с ячейки A1). В данном случае используется свойство `CurrentRegion`, возвращающее объект `Range`, который соответствует прямоугольнику ячеек вокруг заданной ячейки (в данном случае — A1).

```
Sub CopyCurrentRegion2()
   Range("A1").CurrentRegion.Copy Sheets("Лист2").Range("A1")
End Sub
```

Примечание

Использование свойства `CurrentRegion` эквивалентно выбору команды Главная⇨Редактирование⇨Найти и выделить⇨Выделение группы ячеек (Home⇨Editing⇨Find & Select⇨Go To Special) с последующим выбором переключателя Текущую область (Current Region). Еще одно средство — комбинация клавиш <Ctrl+Shift+*>. Чтобы увидеть принцип работы, включите запись макроса и выполните указанные действия. Как правило, значение свойства `CurrentRegion` состоит из прямоугольного диапазона ячеек, окруженных одной или несколькими пустыми строками либо столбцами.

Выделение или определение типов диапазонов

Зачастую работа, выполняемая в VBA, связана с управлением диапазонами — либо выделением диапазона, либо определением диапазона с целью выполнить некоторые действия с ячейками.

Советы по работе с диапазонами

При работе с диапазонами необходимо помнить о следующем.

- Для управления диапазоном его не нужно выделять в программе.
- Если в коде выделяется диапазон, то соответствующий рабочий лист должен быть активным. Для активизации конкретного листа используйте метод `Activate` из коллекции `Worksheets`.
- Функция записи макросов не всегда генерирует самый эффективный код. Зачастую после записи макроса приходится оптимизировать код, чтобы сделать его более эффективным.
- Используйте в программе VBA именованные диапазоны. Например, лучше воспользоваться ссылкой `Range("Total")`, чем `Range("D45")`. Во втором случае при добавлении строки над строкой 45 адрес ячейки изменится. После этого придется изменить макрос, чтобы в нем использовался правильный адрес ячейки (D46).
- Если вы полагаетесь на функцию записи макросов при выборе диапазонов, убедитесь в том, что при записи макроса используются относительные ссылки. Для этого воспользуйтесь командой Разработчик⇨Код⇨Относительные ссылки (Developer⇨ Code⇨Use Relative References).
- Перед выполнением макроса, который управляет каждой ячейкой в текущем диапазоне, рассмотрите ситуацию, когда пользователь выделил столбцы или строки целиком. В большинстве случаев не требуется, чтобы макрос просматривал каждую ячейку выделенного диапазона. На этот случай в макросе должен создаваться новый диапазон, который состоит только из заполненных значениями ячеек (см. раздел "Просмотр выделенного диапазона").
- Excel позволяет выделять несколько несмежных диапазонов. Например, можно выделить диапазон, нажать клавишу <Ctrl>, а затем выделить другой диапазон. Такой диапазон можно протестировать в макросе и определить для него соответствующее действие (см. раздел "Определение типа выделенного диапазона").

Вам необходимо научиться управлять не только свойством `CurrentRegion` (описанным выше), но и методом `End` объекта `Range`. Метод `End` имеет один аргумент, определяющий направление, в котором увеличивается выделение ячеек. Оператор, представленный ниже, выделяет диапазон от активной ячейки до последней непустой ячейки внизу.

```
Range(ActiveCell, ActiveCell.End(xlDown)).Select
```

Как можно догадаться, три остальные константы имитируют комбинации клавиш при выделении в других направлениях: `xlUp` (вверх), `xlToLeft` (влево) и `xlToRight` (вправо).

Предупреждение

Будьте осторожны при работе с методом `End`. Если активная ячейка находится на границе диапазона или диапазон содержит хотя бы одну пустую ячейку, метод `End` может не дать требуемых результатов.

Компакт-диск

На прилагаемом компакт-диске находится рабочая книга (`range selections.xlsm`), в которой определяется несколько распространенных типов выделения ячеек. Открыв ее, вы увидите новое меню — Демонстрация выделения (Selection Demo), — содержащее команды, которые предоставляют пользователю возможность получать различные типы выделений (рис. 11.2).

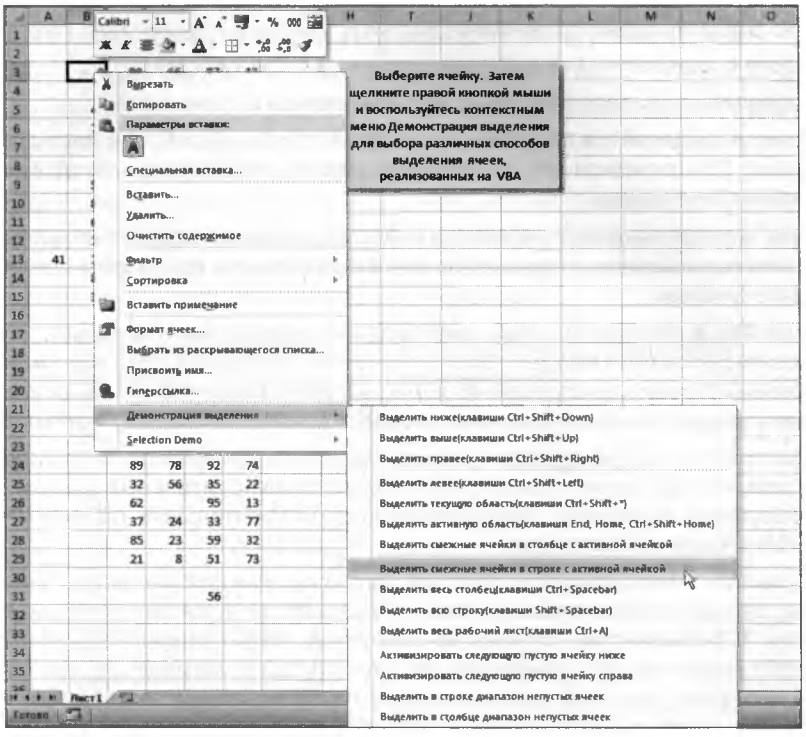


Рис. 11.2. Эта рабочая книга демонстрирует, как выделять диапазоны различной формы в VBA

Запрос значения ячейки

Следующая процедура демонстрирует, как запросить у пользователя значение и вставить его в ячейку A1 активного рабочего листа.

```
Sub GetValue1()
    Range("A1").Value = InputBox("Введите значение")
End Sub
```

На рис. 11.3 показано диалоговое окно для ввода данных.

Однако при выполнении этой процедуры возникает проблема. Если пользователь щелкнет на кнопке **Отмена** (Cancel) в окне ввода данных, то процедура удалит данные, которые находились в текущей ячейке. Модифицированная версия процедуры адекватно реагирует на щелчок на кнопке **Отмена** (Cancel) и не выполняет при этом никаких действий.

```
Sub GetValue2()
    Dim UserEntry As Variant
    UserEntry = InputBox("Введите значение")
    If UserEntry <> "" Then Range("A1").Value = UserEntry
End Sub
```

Во многих случаях следует проверить правильность данных, введенных пользователем. Например, необходимо обеспечить введение только чисел в диапазоне от 1 до 12. Далее представлен способ проверки данных, введенных пользователем. В приведенном ниже примере некорректные данные игнорируются, и окно запроса значения отображается снова. Этот цикл будет повторяться, пока пользователь не введет правильное значение или не щелкнет на кнопке **Отмена** (Cancel).

```
Sub GetValue3()
   Dim UserEntry As Variant
   Dim Msg As String
   Const MinVal As Integer = 1
   Const MaxVal As Integer = 12
   Msg = "Введите значение между " & MinVal & " и " & MaxVal
   Do
      UserEntry = InputBox(Msg)
      If UserEntry = "" Then Exit Sub
      If IsNumeric(UserEntry) Then
         If UserEntry >= MinVal And UserEntry <= MaxVal _
            Then Exit Do
         End If
      Msg = "Вы ввели НЕПРАВИЛЬНОЕ значение."
      Msg = Msg & vbNewLine
      Msg = Msg & "Введите значение от " & _
         MinVal & " и " & MaxVal
   Loop
   ActiveSheet.Range("A1").Value = UserEntry
End Sub
```

Как видно из рис. 11.4, программа также изменяет отображаемое сообщение, если пользователь вводит некорректные данные.

Рис. 11.3. Функция `InputBox` получает от пользователя значение, которое вставляется в ячейку

Рис. 11.4. Проверка данных, введенных пользователем, с помощью функции VBA `InputBox`

Компакт-диск

Три процедуры `GetValue` находятся на прилагаемом к книге компакт-диске в файле `inputbox demo.xlsm`.

Ввод значения в следующую пустую ячейку

Достаточно часто требуется ввести значение в следующую пустую ячейку столбца или строки. В представленном далее примере пользователь должен ввести имя и значение, которые добавляются в следующую пустую строку (рис. 11.5).

```
Sub GetData()
   Dim NextRow As Long
   Dim Entry1 As String, Entry2 As String
Do
' Определение следующей пустой строки
   NextRow = Cells(Rows.Count, 1).End(xlUp).Row + 1

' Запрос на ввод данных
   Entry1 = InputBox("Введите имя")
   If Entry1 = "" Then Exit Sub
      Entry2 = InputBox("Введите количество")
      If Entry2 = "" Then Exit Sub
```

```
'   Запись данных
    Cells(NextRow, 1) = Entry1
    Cells(NextRow, 2) = Entry2
Loop
End Sub
```

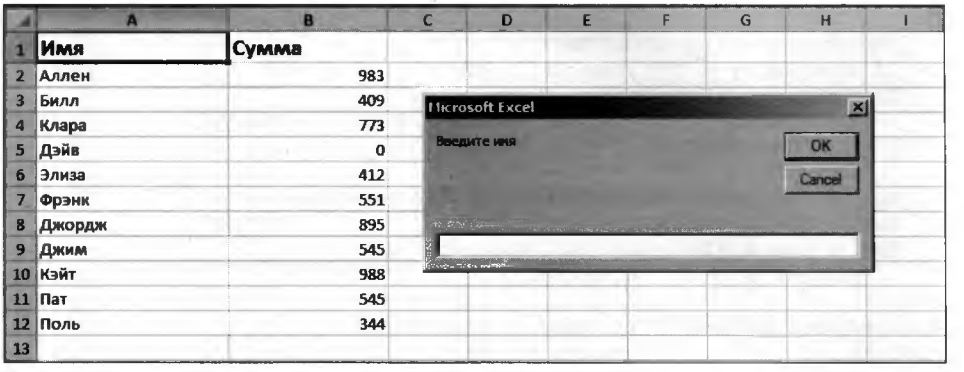


Рис. 11.5. Этот макрос вставляет данные в следующую пустую строку рабочего листа

Чтобы упростить представленную процедуру, проверка данных не выполняется. Обратите внимание, что это бесконечный цикл. Для выхода из него (щелкните на кнопке Cancel (Отмена)) использовались операторы `Exit Sub`.

Компакт-диск

Процедура `GetData` находится на прилагаемом к книге компакт-диске. Соответствующий файл называется `next empty cell.xlsm`.

Обратите внимание на оператор, который определяет значение переменной `NextRow`. Если вам трудно понять принцип его работы, проанализируйте содержимое ячейки: перейдите в последнюю ячейку столбца A и нажмите <End> и <↑>. После этого будет выделена последняя непустая ячейка в столбце A. Свойство `Row` возвращает номер этой строки; чтобы получить расположенную под ней строку (следующую пустую строку), к этому номеру прибавляется 1.

Обратите внимание, что в представленном приеме выделения следующей пустой строки существует один нюанс: если столбец пуст, то следующей пустой строкой будет считаться строка 2. Достаточно легко написать код, учитывающий этот вариант.

Приостановка работы макроса для определения диапазона пользователем

В некоторых ситуациях макрос должен взаимодействовать с пользователем. Например, можно создать макрос, который приостанавливается, когда пользователь указывает диапазон ячеек. Для этого воспользуйтесь функцией Excel `InputBox`.

Примечание

Не путайте метод Excel `InputBox` с функцией VBA `InputBox`. Несмотря на идентичность названий, это далеко не одно и то же.

Процедура, представленная ниже, демонстрирует, как приостановить макрос и разрешить пользователю выбрать ячейку. Затем автоматически формула вставляется в каждую ячейку выделенного диапазона.

```
Sub GetUserRange()
   Dim UserRange As Range

   Prompt = "Выберите диапазон для случайных чисел."
   Title = "Выбор диапазона"
'  Отображение окна ввода данных
   On Error Resume Next
   Set UserRange = Application.InputBox( _
      Prompt:=Prompt, _
      Title:=Title, _
      Default:=ActiveCell.Address, _
      Type:=8) 'Выбор диапазона
   On Error GoTo 0

'  Проверка, была ли нажата кнопка отмены?
   If UserRange Is Nothing Then
      MsgBox "Действие отменено."
   Else
      UserRange.Formula = "=RAND()"
   End If
End Sub
```

Окно ввода данных показано на рис. 11.6.

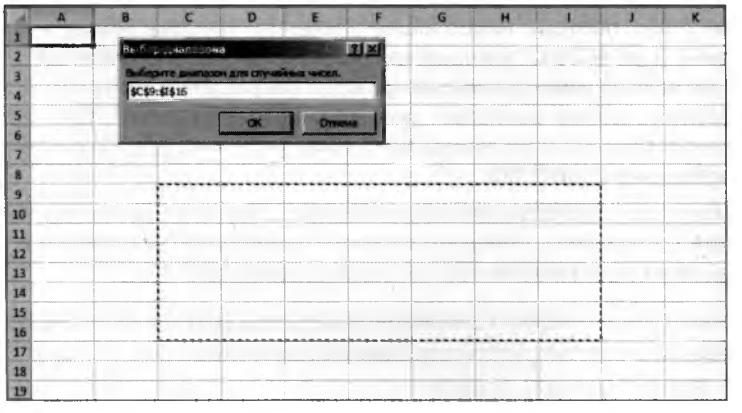


Рис. 11.6. Использование окна ввода данных с целью приостановки выполнения макроса

Компакт-диск

Этот пример под названием `prompt for a range.xlsm` можно найти на прилагаемом к книге компакт-диске.

Ключевым моментом в этой процедуре является определение аргумента `Type` равным 8. Кроме того, обратите внимание на использование оператора `On Error Resume Next`. Он игнорирует ошибку, происходящую по вине пользователя, который щелкает на кнопке Отмена (Cancel). В таком случае объектная переменная `UserRange` не получает значения. В данном примере отображается окно сообщения с текстом "Отменено". Если

же пользователь щелкнет на кнопке OK, то макрос продолжит выполняться. Строка `On Error Go To` указывает на переход к стандартной обработке ошибки.

Проверка корректного выделения диапазона необязательна. Excel позаботится об этом вместо вас.

Предупреждение

Обязательно проверьте, включено ли обновление экрана при использовании метода `InputBox` для выделения диапазона. Если обновление экрана отключено, вы не сможете выделить рабочий лист. Чтобы проконтролировать обновление экрана, в процессе выполнения макроса используйте свойство `ScreenUpdating` объекта `Application`.

Подсчет выделенных ячеек

Работая с макросом, который обрабатывает выделенный диапазон ячеек, можно использовать свойство `Count`, чтобы определить, сколько ячеек содержится в выделенном (или любом другом) диапазоне. Например, следующий оператор демонстрирует окно сообщения, которое отображает количество ячеек в текущем выделенном диапазоне:

```
MsgBox Selection.Count
```

Предупреждение

Поскольку размер рабочего листа в Excel 2010 был сильно увеличен, свойство `Count` может генерировать ошибку. Свойство `Count` использует тип данных `Long`, поэтому наибольшее значение, которое может храниться здесь, равно 2147483647. Например, если пользователь выделил 2048 столбцов (2147483648 ячеек), свойство `Count` сгенерирует ошибку. К счастью, Microsoft добавила новое свойство, называемое `CountLarge`. Оно использует тип данных `Double`, причем максимальная величина обрабатываемого значения составляет 1,79+E^308.

В большинстве случаев можно воспользоваться свойством `Count`. Если же нужно подсчитать очень много ячеек (например, все ячейки в рабочем листе), используйте свойство `CountLarge` вместо `Count`.

Если активный лист содержит диапазон `data`, то следующий оператор присваивает количество ячеек в диапазоне `data` переменной с названием `CellCount`.

```
CellCount = Range("data").Count
```

Вы можете также определить, сколько строк или столбцов содержится в диапазоне. Следующее выражение вычисляет количество столбцов в выделенном в данный момент диапазоне:

```
Selection.Columns.Count
```

Для определения количества строк в диапазоне вы также можете использовать свойство `Rows`. Следующий оператор пересчитывает количество строк в диапазоне с названием `data` и присваивает это количество переменной `RowCount`.

```
RowCount = Range("data").Rows.Count
```

Определение типа выделенного диапазона

В Excel можно выделить следующие типы диапазонов:

- отдельная ячейка;
- смежный диапазон ячеек;
- один или более полных столбцов;
- одна или более полных строк;
- весь рабочий лист;
- комбинация перечисленных выше типов (называемая множественным выделением).

В результате, когда процедура VBA обрабатывает выделенный диапазон, нельзя исходить из каких-либо предположений о типе этого диапазона.

В случае множественного выделения объект `Range` включает несмежные области. Чтобы определить, является ли выделение множественным, используется метод `Areas`, возвращающий коллекцию `Areas`. Эта коллекция представляет все диапазоны в множественном выделении.

Чтобы определить, содержатся ли в выделенном диапазоне множественные области, примените выражение, подобное следующему:

```
NumAreas = Selection.Areas.Count
```

Если переменная `NumAreas` содержит значение, которое больше единицы, то выделение является множественным.

Ниже приводится код функции `AreaType`, которая возвращает текстовую строку, описывающую тип выделенного диапазона.

```
Function AreaType(RangeArea As Range) As String
' Возвращает тип диапазона
  Select Case True
     Case RangeArea.Cells.CountLarge = 1
        AreaType = "Ячейка"
     Case RangeArea.CountLarge = Cells.CountLarge
        AreaType = "Рабочий лист"
     Case RangeArea.Rows.Count = Cells.Rows.Count
        AreaType = "Столбец"
     Case RangeArea.Columns.Count = Cells.Columns.Count
        AreaType = "Строка"
     Case Else
        AreaType = "Блок"
  End Select
End Function
```

Эта функция получает в качестве аргумента объект `Range` и возвращает одну из пяти строк, описывающих данную область: ячейку (`Cell`), рабочий лист (`Worksheet`), столбец (`Column`), строку (`Row`) или блок (`Block`). Функция использует конструкцию `Select Case` для определения одного из четырех выражений сравнения, которое имеет значение `True`. Например, если диапазон состоит из одной ячейки, функция возвращает значение `ячейка`. Если количество ячеек в диапазоне равно количеству ячеек в рабочем листе, то функция возвращает `рабочий лист`. Если количество строк в диапазоне равно количеству строк в рабочем листе, функция возвращает `столбец`. Если количество столбцов в диапазоне равно количеству столбцов на рабочем листе, функция возвращает `строка`. Если ни одно из выражений `Case` не равно `True`, то функция возвращает строку `блок`.

Обратите внимание, что для подсчета количества ячеек использовалось свойство `CountLarge`. Как уже отмечалось, количество выделенных ячеек может превышать предел для свойства `Count`.

Компакт-диск

Рассматриваемый пример находится на прилагаемом к книге компакт-диске (файл `about range selection.xlsm`). Рабочая книга включает процедуру (`RangeDescription`), использующую функцию `AreaType` для отображения окна сообщения, в котором описывается текущий выделенный диапазон. На рис. 11.7 показан пример. Изучив принцип работы данной процедуры, вы сможете эффективно работать с объектами `Range`.

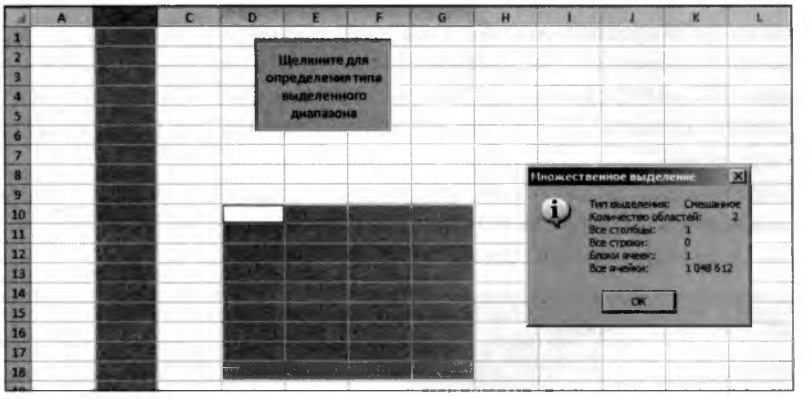


Рис. 11.7. Процедура VBA анализирует текущий выделенный диапазон

Примечание

Возможно, вас удивит тот факт, что Excel обеспечивает создание идентичных множественных выделений. Например, если, удерживая клавишу <Ctrl>, пять раз щелкнуть на ячейке A1, выделение будет содержать пять идентичных объектов. То же самое относится к процедуре `RangeDescription`, которая различает идентичные ячейки.

Просмотр выделенного диапазона

Вы можете столкнуться с трудностями при создании макроса, который оценивает каждую ячейку в диапазоне и выполняет операцию, определенную заданному критерию. Ниже приведена процедура, которая представляет собой пример подобного макроса. Процедура `ColorNegative` устанавливает красный фоновый цвет для ячеек, которые содержат отрицательные значения. Цвет фона для других ячеек не определяется.

Примечание

Этот пример приводится исключительно в учебных целях. На практике лучше использовать условное форматирование Excel.

```
Sub ColorNegative()
'  Определяет красный цвет для ячеек с отрицательными
   значениями
   Dim cell As Range
   If TypeName(Selection) <> "Range" Then Exit Sub
```

```
    Application.ScreenUpdating = False
    For Each cell In Selection
       If cell.Value < 0 Then
          cell.Interior.Color = RGB(255, 0, 0)
       Else
          cell.Interior.Color = xlNone
       End If
    Next cell
End Sub
```

Процедура `ColorNegative` работает, но в ней имеется серьезный недостаток. К примеру, какой результат вы получите, если выделение состоит из полного столбца? Или десяти столбцов? Или всего рабочего листа? Пользователь, скорее всего, уснет раньше, чем будут оценены все ячейки.

Ниже представлено наилучшее решение (`ColorNegative2`). В этой переработанной процедуре создается объектная переменная `WorkRange` типа `Range`, которая представляет собой пересечение выделенного диапазона и диапазона рабочего листа. На рис. 11.8 показан пример выделения целого столбца D, включающего 1048576 ячеек. Диапазон, используемый рабочим листом, включает ячейки B2:I16. В результате пересечения этих диапазонов получаем область D2:D16, которая намного меньше исходного выделенного диапазона. Естественно, время, затрачиваемое на обработку 15 ячеек, намного меньше времени, уходящего на обработку 1048576 ячеек.

	A	B	C	D	E	F	G	H	I
1									
2		-5	0	-7	3	-3	7	-6	-9
3		-5	-6	-6	-10	-1	10	9	-10
4		-2	5	1	4	-3	3	-8	-3
5		1	8	-3	-8	1	8	8	6
6		0	-4	-3	3	-1	7	5	2
7		-10	4	1	8	1	-8	7	9
8		5	-4	-1	7	10	-1	8	-3
9		1	4	1	-8	-2	-1	-6	8
10		-8	-3	10	-1	7	6	7	9
11		0	-2	-2	-1	9	7	7	7
12		10	4	7	6	10	-10	10	4
13		-5	-1	9	7	0	8	6	9
14		3	-4	10	-10	9	-9	2	-4
15		4	9	0	8	4	7	-1	-4
16		0	1	9	-9	2	7	-7	0
17									
18									
19									

Рис. 11.8. В результате пересечения используемого диапазона и выделенного диапазона рабочего листа уменьшается количество обрабатываемых ячеек

```
Sub ColorNegative2()
' Определяет красный цвет для ячеек с отрицательными значениями
  Dim WorkRange As Range
  Dim cell As Range
  If TypeName(Selection) <> "Range" Then Exit Sub
  Application.ScreenUpdating = False
  Set WorkRange = Application.Intersect(Selection, _
     ActiveSheet.UsedRange)
  For Each cell In WorkRange
```

```
      If cell.Value < 0 Then
         cell.Interior.Color = RGB(255, 0, 0)
      Else
         cell.Interior.Color = xlNone
      End If
   Next cell
End Sub
```

Процедура ColorNegative2 представляет собой результат переработки предыдущей процедуры, но все же она недостаточно эффективна, поскольку обрабатывает пустые ячейки. Поэтому появилась третья версия, ColorNegative3, которая немного больше по размеру, но намного эффективнее. В ней используется метод SpecialCells, с помощью которого генерируются два поднабора выделенной области: один поднабор (ConstantCells) включает ячейки, которые содержат исключительно числовые константы; второй поднабор (FormulaCells) включает ячейки, содержащие числовые формулы. Обработка ячеек в этих поднаборах осуществляется с помощью двух конструкций For Each-Next. Благодаря тому, что исключается обработка пустых и нетекстовых ячеек, скорость выполнения макроса существенно увеличивается.

```
Sub ColorNegative3()
'  Определяет красный цвет для ячеек с отрицательными значениями
   Dim FormulaCells As Range, ConstantCells As Range
   Dim cell As Range
   If TypeName(Selection) <> "Range" Then Exit Sub
   Application.ScreenUpdating = False

'  Создание поднаборов на основе исходного выделения
   On Error Resume Next
   Set FormulaCells = Selection.SpecialCells(xlFormulas, xlNumbers)
   Set ConstantCells = Selection.SpecialCells(xlConstants, lNumbers)
   On Error GoTo 0

'  Обработка ячеек формулы
   If Not FormulaCells Is Nothing Then
      For Each cell In FormulaCells
         If cell.Value < 0 Then
            cell.Interior.Color = RGB(255, 0, 0)
         Else
            cell.Interior.Color = xlNone
         End If
      Next cell
   End If
'  Обработка ячеек, содержащих константы
   If Not ConstantCells Is Nothing Then
      For Each cell In ConstantCells
         If cell.Value < 0 Then
            cell.Interior.Color = RGB(255, 0, 0)
         Else
            cell.Interior.Color = xlNone
         End If
      Next cell
   End If
End Sub
```

Примечание

Оператор On Error необходим, поскольку метод SpecialCells генерирует ошибку, если не находит в диапазоне ячеек указанного типа.

Компакт-диск

Рабочая книга, содержащая три процедуры ColorNegative, находится на прилагаемом к книге компакт-диске и называется efficient looping.xlsm.

Удаление всех пустых строк

Следующая процедура удаляет все пустые строки в активном рабочем листе. Она достаточно эффективна, так как не проверяет все без исключения строки, а просматривает только строки в так называемом "используемом диапазоне", определяемом с помощью свойства UsedRange объекта Worksheet.

```
Sub DeleteEmptyRows()
   Dim LastRow As Long
   Dim r As Long
   Dim Counter As Long
   Application.ScreenUpdating = False
   LastRow = ActiveSheet.UsedRange.Rows.Count + _
      ActiveSheet.UsedRange.Rows(1).Row - 1
   For r = LastRow To 1 Step -1
      If Application.WorksheetFunction.CountA(Rows(r)) = 0 Then
         Rows(r).Delete
         Counter = Counter + 1
      End If
   Next r
   Application.ScreenUpdating = True
   MsgBox Counter & " пустые строки удалены."
End Sub
```

Первый шаг — определить последнюю используемую строку и присвоить этот номер строки переменной LastRow. Это не так просто, как можно ожидать, поскольку текущий диапазон необязательно начинается со строки 1. Следовательно, значение LastRow вычисляется таким образом: к найденному количеству строк используемого диапазона прибавляется номер первой строки текущего диапазона и вычитается 1.

В процедуре применена функция Excel СЧЁТЗ (COUNTA), определяющая, является ли строка пустой. Если данная функция для конкретной строки возвращает 0, то эта строка пустая. Обратите внимание, что процедура просматривает строки снизу вверх и использует отрицательное значение шага в цикле For-Next. Это необходимо, поскольку при удалении все последующие строки перемещаются "вверх" в рабочем листе. Если бы в цикле просмотр выполнялся сверху вниз, то значение счетчика цикла после удаления строки оказалось бы неправильным.

В макросе используется еще одна переменная, Counter, с помощью которой подсчитывается количество удаленных строк. Эта величина отображается в окне сообщения по завершении процедуры.

Компакт-диск

Рабочая книга, содержащая описанный выше пример, находится на прилагаемом к книге компакт-диске и называется delete empty rows.xlsm.

Дублирование строк

Пример, рассматриваемый в этом разделе, демонстрирует использование возможностей VBA для создания дубликатов строк. На рис. 11.9 показан пример рабочего листа,

используемого организаторами лотереи. В столбце A вводится имя. В столбце B содержится количество лотерейных билетов, приобретенных одним покупателем. В столбце C находится случайное число (сгенерированное с помощью функции СЛЧИС (RAND)). Победитель определяется путем сортировки данных в третьем столбце (выигрыш соответствует наибольшему случайному числу).

	A	B	C
1	Имя	Количество билетов	Случайное число
2	Алан	1	0,276807291
3	Барбара	2	0,730227266
4	Чарли	1	0,946701186
5	Дэйв	5	0,015892566
6	Фрэнк	3	0,678673514
7	Гильда	1	0,554946703
8	Губерт	1	0,467700824
9	Инес	2	0,830401846
10	Марк	1	0,822189002
11	Нора	10	0,371007999
12	Пенелопа	2	0,918587635
13	Рэнси	1	0,172321775
14	Уэнди	2	0,909526052
15			

Рис. 11.9. Дублирование строк на основе значений в столбце B

А теперь нужно продублировать строки, в результате чего количество строк для каждого участника лотереи будут соответствовать количеству купленных им билетов. Например, если Барбара приобрела два билета, для нее создаются две строки. Ниже показана процедура, выполняющая вставку новых строк.

```
Sub DupeRows()
   Dim cell As Range
'  В первой ячейке указано количество билетов
   Set cell = Range("B2")
   Do While Not IsEmpty(cell)
      If cell > 1 Then
         Range(cell.Offset(1, 0), cell.Offset(cell.Value - 1, _
            0)).EntireRow.Insert
         Range(cell, cell.Offset(cell.Value - 1, _
            1)).EntireRow.FillDown
      End If
      Set cell = cell.Offset(cell.Value, 0)
   Loop
End Sub
```

Объектная переменная `cell` была инициализирована ячейкой B2, первой ячейкой, в которой находится числовая величина. Вставка новых строк осуществляется в цикле, а их копирование происходит с помощью метода `FillDown`. Значение переменной `cell` увеличивается на единицу, после чего выбирается следующий участник лотереи. Цикл выполняется до тех пор, пока не встретится пустая ячейка. На рис. 11.10 показан рабочий лист после выполнения этой процедуры.

Компакт-диск

Рабочая книга, содержащая описанный выше пример, находится на прилагаемом компакт-диске в файле `duplicate rows.xlsm`.

Определение диапазона, находящегося в другом диапазоне

Функция InRange, показанная ниже, имеет два аргумента, оба — объекты Range. Функция возвращает значение True (Истина), если первый диапазон содержится во втором.

```
Function InRange(rng1, rng2) As Boolean
' Возвращает значение True, если диапазон
rng1 входит в диапазон rng2
   InRange = False
   If rng1.Parent.Parent.Name =
rng2.Parent.Parent.Name Then
      If rng1.Parent.Name =
rng2.Parent.Name Then
         If Union(rng1, rng2).Address =
rng2.Address Then
            InRange = True
         End If
      End If
   End If
End Function
```

Возможно, функция InRange кажется сложнее, чем того требует ситуация, поскольку в коде должна быть реализована проверка принадлежности двух диапазонов одной и той же книге и рабочему листу. Обратите внимание, что в процедуре используется свойство Parent, которое возвращает объект-контейнер заданного объекта. Например, следующее выражение возвращает название листа для объекта rng1:

```
rng1.Parent.Name
```

Следующее выражение возвращает название рабочей книги rng1:

```
rng1.Parent.Parent.Name
```

	A	B	C
1	Имя	Количество билетов	Случайное число
2	Алан	1	0,811712817
3	Барбара	2	0,408809459
4	Барбара	2	0,817294658
5	Чарли	1	0,965809492
6	Дэйв	5	0,777591092
7	Дэйв	5	0,704535562
8	Дэйв	5	0,566578818
9	Дэйв	5	0,560431236
10	Дэйв	5	0,849621677
11	Фрэнк	3	0,349464908
12	Фрэнк	3	0,347134158
13	Фрэнк	3	0,33883254
14	Гильда	1	0,519896123
15	Губерт	1	0,956426929
16	Инес	2	0,108305391
17	Инес	2	0,949941449
18	Марк	1	0,563116763
19	Нора	10	0,881752223
20	Нора	10	0,588564786
21	Нора	10	0,025840496
22	Нора	10	0,199440989
23	Нора	10	0,067126822
24	Нора	10	0,54658458
25	Нора	10	0,293597337
26	Нора	10	0,936284654
27	Нора	10	0,975085451
28	Нора	10	0,480195995
29	Пенелопа	2	0,073935047
30	Пенелопа	2	0,192292217
31	Рэнси	1	0,10695783
32	Уэнди	2	0,896294737
33	Уэнди	2	0,486119769
34			

Рис. 11.10. В соответствии со значением в столбце B добавлены новые строки

Функция VBA Union возвращает объект Range, который представляет собой объединение двух объектов типа Range. Объединение содержит все ячейки, относящиеся к исходным диапазонам. Если адрес объединения двух диапазонов совпадает с адресом второго диапазона, первый диапазон входит в состав второго диапазона.

Компакт-диск

Рабочая книга, включающая только что описанную функцию, находится на прилагаемом компакт-диске в файле inrange function.xlsm.

Определение типа данных ячейки

В состав Excel входит ряд встроенных функций, которые могут помочь определить тип данных, содержащихся в ячейке. Это функции ЕНЕТЕКСТ (ISTEXT), ЕЛОГИЧ (ISLOGICAL) и ЕОШИБКА (ISERROR). Кроме того, VBA поддерживает функции IsEmpty, IsDate и IsNumeric.

Ниже описана функция CellType, которая принимает аргумент-диапазон и возвращает строку (Пусто, Текст, Булево выражение, Ошибка, Дата, Время или Значение), описывающую тип данных левой верхней ячейки этого диапазона. Такую функцию можно использовать в формуле рабочего листа или вызвать из другой процедуры VBA.

```
Function CellType(Rng)
'  Возвращает тип верхней левой
'  ячейки в диапазоне
   Dim TheCell As Range
   Set TheCell = Rng.Range("A1")
   Select Case True
      Case IsEmpty(TheCell)
         CellType = "Пусто"
      Case Application.IsText(TheCell)
         CellType = "Текст"
      Case Application.IsLogical(TheCell)
         CellType = "Булево выражение"
      Case Application.IsErr(TheCell)
         CellType = "Error"
      Case IsDate(TheCell)
         CellType = "Дата"
      Case InStr(1, TheCell.Text, ":") <> 0
         CellType = "Время"
      Case IsNumeric(TheCell)
         CellType = "Значение"
   End Select
End Function
```

Обратите внимание на использование оператора Set TheCell. Функция CellType получает аргумент-диапазон произвольного размера, но этот оператор указывает, что функция оперирует только левой верхней ячейкой диапазона (представленной переменной TheCell).

Компакт-диск

Рабочая книга, содержащая описанную выше функцию, находится на прилагаемом компакт-диске в файле celltype function.xlsm.

Чтение и запись диапазонов

Многие задачи, выполняемые в электронных таблицах, связаны с переносом значений из массива в диапазон ячеек или из диапазона в массив. Следует отметить, что Excel получает данные из диапазонов значительно быстрее, чем записывает их. Процедура WriteReadRange демонстрирует относительную скорость записи и чтения диапазона.

Процедура создает массив и затем использует циклы For-Next для записи этого массива в диапазон и считывания данных обратно в массив. С помощью функции Timer вычисляется время, необходимое для каждой операции.

```
Sub WriteReadRange()
   Dim MyArray()
   Dim Time1 As Double
   Dim NumElements As Long, i As Long
   Dim WriteTime As String, ReadTime As String
   Dim Msg As String
   NumElements = 60000
   ReDim MyArray(1 To NumElements)
```

```
' Заполнение массива
  For i = 1 To NumElements
     MyArray(i) = i
  Next i

' Запись массива в диапазон
  Time1 = Timer
  For i = 1 To NumElements
     Cells(i, 1) = MyArray(i)
  Next i
  WriteTime = Format(Timer - Time1, "00:00")

' Считывание диапазона в массив
  Time1 = Timer
  For i = 1 To NumElements
     MyArray(i) = Cells(i, 1)
  Next i
  ReadTime = Format(Timer - Time1, "00:00")

' Отображение результатов
  Msg = "Запись: " & WriteTime
  Msg = Msg & vbCrLf
  Msg = Msg & "Чтение: " & ReadTime
  MsgBox Msg, vbOKOnly, NumElements & " элементов"
End Sub
```

В моей системе для записи массива из 60 тысяч элементов в диапазон потребовалось 58 секунд, и только 1 секунда ушла на то, чтобы занести этот диапазон обратно в массив.

Более эффективный способ записи в диапазон

В примере из предыдущего раздела для перемещения содержимого массива в диапазон используется цикл `For-Next`. В данном разделе показан более эффективный способ выполнения этой операции.

Начнем со следующего примера, в котором продемонстрирован наиболее очевидный (но не самый эффективный) способ заполнения диапазона. В этом примере для вставки значений в диапазон используется цикл `For-Next`.

```
Sub LoopFillRange()
' Заполнение диапазона в цикле по ячейкам
  Dim CellsDown As Long, CellsAcross As Integer
  Dim CurrRow As Long, CurrCol As Integer
  Dim StartTime As Double
  Dim CurrVal As Long

' Получение размеров
  CellsDown = InputBox("Сколько ячеек в высоту?")
  If CellsDown = 0 Then Exit Sub
  CellsAcross = InputBox("Сколько ячеек в ширину?")
  If CellsAcross = 0 Then Exit Sub

' Запись момента начала
  StartTime = Timer

' Просмотр ячеек и вставка значений
  CurrVal = 1
  Application.ScreenUpdating = False
  For CurrRow = 1 To CellsDown
```

```
      For CurrCol = 1 To CellsAcross
         ActiveCell.Offset(CurrRow - 1, _
            CurrCol - 1).Value = CurrVal
         CurrVal = CurrVal + 1
      Next CurrCol
   Next CurrRow

'  Отображение времени выполнения операции
   Application.ScreenUpdating = True
   MsgBox Format(Timer - StartTime, "00.00") & " секунд"
End Sub
```

Следующий пример демонстрирует самый эффективный способ получения того же результата. Программа вставляет значения в массив и использует всего один оператор для перенесения содержимого массива в диапазон.

```
Sub ArrayFillRange()
'  Заполнение диапазона путем перенесения массива
   Dim CellsDown As Long, CellsAcross As Integer
   Dim i As Long, j As Integer
   Dim StartTime As Double
   Dim TempArray() As Long
   Dim TheRange As Range
   Dim CurrVal As Long
'  Получение размеров
   CellsDown = InputBox("Сколько ячеек в высоту?")
   If CellsDown = 0 Then Exit Sub
   CellsAcross = InputBox("Сколько ячеек в ширину?")
   If CellsAcross = 0 Then Exit Sub
'  Запись момента начала
   StartTime = Timer
'  Изменение размера временного массива
   ReDim TempArray(1 To CellsDown, 1 To CellsAcross)
'  Определение диапазона на рабочем листе
   Set TheRange = ActiveCell.Range(Cells(1, 1), _
      Cells(CellsDown, CellsAcross))
'  Заполнение временного массива
   CurrVal = 0
   Application.ScreenUpdating = False
   For i = 1 To CellsDown
      For j = 1 To CellsAcross
         TempArray(i, j) = CurrVal + 1
         CurrVal = CurrVal + 1
      Next j
   Next i
'  Перенесение временного массива в рабочую книгу
   TheRange.Value = TempArray
'  Отображение времени вычислений
   Application.ScreenUpdating = True
   MsgBox Format(Timer - StartTime, "00.00") & " секунд"
End Sub
```

Например, в моей системе на заполнение массива размером 1000×250 ячеек (250 тысяч ячеек) методом цикла уходит 10,05 секунды. Метод перенесения массива потребовал только 0,18 секунды для получения тех же самых результатов — более чем в 50 раз быстрее! Мораль? Если необходимо переносить большие объемы данных на лист Excel, по возможности избегайте циклов.

Примечание

Время вычисления сильно зависит от наличия формул. В общем случае время перенесения массива будет минимальным при использовании рабочих книг, в которых отсутствуют формулы либо выбран режим пересчета формул Вручную.

Компакт-диск

Рабочая книга, включающая процедуры `WriteReadRange`, `LoopFillRange` и `ArrayFillRange`, находится на прилагаемом компакт-диске в файле `loop vs array fill range.xlsm`.

Перенесение одномерных массивов

В примере из предыдущего раздела рассматривался двухмерный массив, который можно использовать для управления прямоугольными диапазонами (содержащими несколько строк и столбцов).

При перенесении одномерного массива диапазон должен быть горизонтальным, т.е. это должна быть строка длительностью в несколько *столбцов*. Если же необходимо использовать вертикальный диапазон, сначала следует транспонировать массив. Для этого используйте функцию Excel `ТРАНСП` (TRANSPOSE). В следующем примере 100-элементный массив вставляется в вертикальный диапазон на рабочем листе (A1:A100).

```
Range("A1:A100").Value = _
  Application.WorksheetFunction.Transpose(MyArray)
```

Предупреждение

Функция Excel `ТРАНСП` не работает с массивами, которые содержат более 65536 элементов.

Перенесение диапазона в массив типа `Variant`

В настоящем разделе рассматривается еще один способ управления данными Excel в VBA. В примере, показанном ниже, диапазон ячеек переносится в двухмерный массив типа `Variant`. Затем в окнах сообщений отображаются границы обеих размерностей массива.

```
Sub RangeToVariant()
  Dim x As Variant
  x = Range("A1:L600").Value
  MsgBox UBound(x, 1)
  MsgBox UBound(x, 2)
End Sub
```

В данном примере в первом окне сообщения отображается `600` (количество строк в массиве), а во втором окне сообщения — `12` (количество столбцов). Вы увидите, что перенесение данных диапазона в массив типа `Variant` происходит за долю секунды.

В следующем примере считывается диапазон в массив `Variant`, выполняется простая операция умножения над каждым элементом массива и массив перемещается обратно в диапазон.

```
Sub RangeToVariant2()
  Dim x As Variant
  Dim r As Long, c As Integer
' Чтение данных
```

```
   x = Range("data").Value

'  Просмотр массива
   For r = 1 To UBound(x, 1)
      For c = 1 To UBound(x, 2)
'        Умножение на 2
         x(r, c) = x(r, c) * 2
      Next c
   Next r
'  Передача переменной типа Variant обратно на лист
   Range("data") = x
End Sub
```

Данная процедура работает очень быстро.

Компакт-диск

Рабочая книга, содержащая этот пример, находится на прилагаемом компакт-диске в файле `variant transfer.xlsm`.

Выбор ячеек по значению

Пример из этого раздела демонстрирует выделение ячеек на основе их значений. Довольно странно, что в Excel отсутствует метод осуществления этой операции. Эта проблема решается с помощью разработанной мною процедуры `SelectByValue`. Данная процедура выделяет ячейки с отрицательными значениями, но это поведение можно легко изменить.

```
Sub SelectByValue()
   Dim Cell As Object
   Dim FoundCells As Range
   Dim WorkRange As Range

   If TypeName(Selection) <> "Range" Then Exit Sub

'  Проверить все или выделенное?
   If Selection.CountLarge = 1 Then
      Set WorkRange = ActiveSheet.UsedRange
   Else
      Set WorkRange = Application.Intersect(Selection, _
         ActiveSheet.UsedRange)
   End If

'  Ограничение поиска только числовыми ячейками
   On Error Resume Next
   Set WorkRange = WorkRange.SpecialCells(xlConstants, xlNumbers)
   If WorkRange Is Nothing Then Exit Sub
   On Error GoTo 0

'  Обход каждой ячейки, добавление в диапазон FoundCells ячеек,
'  удовлетворяющих заданным условиям
   For Each Cell In WorkRange
      If Cell.Value < 0 Then
         If FoundCells Is Nothing Then
            Set FoundCells = Cell
         Else
            Set FoundCells = Union(FoundCells, Cell)
         End If
```

```
    End If
  Next Cell
' Отображение сообщения либо выделение ячеек
  If FoundCells Is Nothing Then
    MsgBox "Не найдены ячейки, соответствующие заданным условиям."
  Else
    FoundCells.Select
  End If
End Sub
```

Процедура начинается с проверки выделенной области. Если выделена одна ячейка, поиск выполняется во всей рабочей книге. Если в выделенную область включаются хотя бы две ячейки, поиск осуществляется только в выделенном диапазоне. Уточнение результатов поиска в диапазоне производится с помощью метода `SpecialCells`, причем создается объект `Range`, который содержит только числовые константы.

Код в цикле `For-Next` проверяет значение ячейки. Если оно соответствует критерию (меньше 0), ячейка добавляется в объект `FoundCells Range` с помощью метода `Union`. Обратите внимание, что для первой ячейки метод `Union` неприменим. Если диапазон `FoundCells` не содержит ячеек, попытка применения метода `Union` приведет к появлению ошибки. Таким образом код проверяет, не будет ли значение объекта `FoundCells` равно `Nothing`.

По завершении цикла объект `FoundCells` будет состоять из ячеек, которые удовлетворяют критерию (либо получит значение `Nothing`, если ячейки не найдены). Если ячейки не найдены, отображается окно сообщения. В противном случае выделяются ячейки.

Компакт-диск

Рассмотренный выше пример находится на прилагаемом к книге компакт-диске в файле `select by value.xlsm`.

Копирование несмежных диапазонов

Если вы попытаетесь копировать несмежные (или, как их еще называют, несвязные) выделенные диапазоны, ничего из этого не выйдет, поскольку Excel не поддерживает эти операции. Подобная попытка завершится отображением сообщения об ошибке `Данная команда неприменима для несвязных диапазонов` (That command cannot be used on multiple selections).

Как правило, любое ограничение в Excel можно преодолеть, создав специальный макрос. Ниже приведена процедура VBA, обеспечивающая копирование нескольких выделенных областей в другое место.

```
Sub CopyMultipleSelection()
  Dim SelAreas() As Range
  Dim PasteRange As Range
  Dim UpperLeft As Range
  Dim NumAreas As Long, i As Long
  Dim TopRow As Long, LeftCol As Long
  Dim RowOffset As Long, ColOffset As Long

  If TypeName(Selection) <> "Range" Then Exit Sub

' Хранение областей в виде отдельных объектов Range
  NumAreas = Selection.Areas.Count
  ReDim SelAreas(1 To NumAreas)
```

```
   For i = 1 To NumAreas
      Set SelAreas(i) = Selection.Areas(i)
   Next
'  Определение верхней левой ячейки в множественном выделении
   TopRow = ActiveSheet.Rows.Count
   LeftCol = ActiveSheet.Columns.Count
   For i = 1 To NumAreas
      If SelAreas(i).Row < TopRow Then TopRow = SelAreas(i).Row
      If SelAreas(i).Column < LeftCol Then LeftCol = SelAreas _
         (i).Column
   Next
   Set UpperLeft = Cells(TopRow, LeftCol)

'  Получить адрес вставки
   On Error Resume NextSet PasteRange = Application.InputBox _
      (Prompt:="Указание верхней левой ячейки для диапазона _
      вставки:", Title:="Копирование множественного _
      выделения ", Type:=8)
   On Error GoTo 0
'  Выход после отмены операции
   If TypeName(PasteRange) <> "Range" Then Exit Sub

'  Убедитесь в том, что используется только верхняя левая ячейка
   Set PasteRange = PasteRange.Range("A1")

'  Копирование и вставка каждой области
   For i = 1 To NumAreas
      RowOffset = SelAreas(i).Row - TopRow
      ColOffset = SelAreas(i).Column - LeftCol
      SelAreas(i).Copy PasteRange.Offset(RowOffset, ColOffset)
   Next i
End Sub
```

На рис. 11.11 показан запрос на выбор цели копирования.

Компакт-диск

На прилагаемом компакт-диске находятся рабочая книга с приведенным выше примером, а также еще одна версия, которая отображает предупреждение в случае опасности перезаписи данных. Соответствующий файл называется `copy multiple selection.xlsm`.

Управление рабочими книгами и листами

Приводимые в этом разделе примеры демонстрируют различные способы использования VBA для управления рабочими книгами и листами.

Сохранение всех рабочих книг

Следующая процедура циклически просматривает все рабочие книги в коллекции `Workbooks` и сохраняет каждый файл, который сохранялся ранее.

```
Public Sub SaveAllWorkbooks()
   Dim Book As Workbook
   For Each Book In Workbooks
      If Book.Path <> "" Then Book.Save
```

```
   Next Book
End Sub
```

Обратите внимание на то, как используется свойство `Path`. Если для какой-либо рабочей книги свойство `Path` не задано, значит, файл еще не сохранялся (это новая рабочая книга). Данная процедура игнорирует такие рабочие книги и сохраняет только те из них, свойство `Path` которых имеет ненулевое значение.

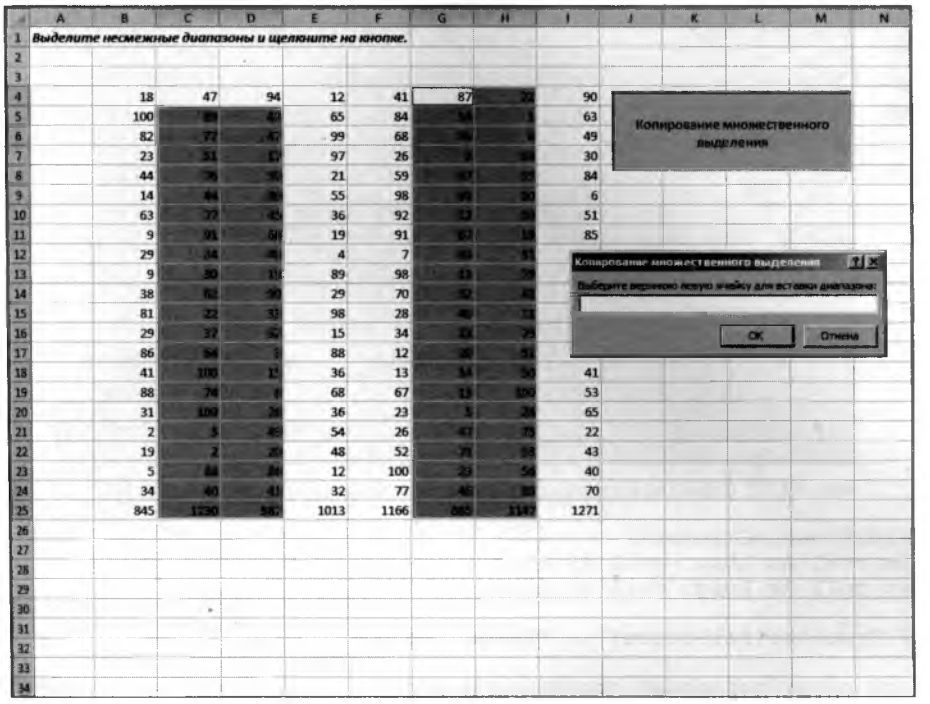


Рис. 11.11. Использование метода Excel `InputBox` для запроса местоположения ячейки

Сохранение и закрытие всех рабочих книг

Следующая процедура циклически просматривает коллекцию `Workbooks`. Программа сохраняет и закрывает все рабочие книги.

```
Sub CloseAllWorkbooks()
   Dim Book As Workbook
   For Each Book In Workbooks
      If Book.Name <> ThisWorkbook.Name Then
         Book.Close savechanges:=True
      End If
   Next Book
   ThisWorkbook.Close savechanges:=True
End Sub
```

Обратите внимание, что процедура использует оператор `If`, чтобы определить, содержит ли данная рабочая книга текущий выполняемый код. Это необходимо, так как при закрытии рабочей книги, содержащей процедуру, программа автоматически завершает свое выполнение, причем остальные рабочие книги не будут сохранены и закрыты.

Частичное сокрытие элементов рабочего листа

В примере из этого раздела скрываются все строки и столбцы рабочего листа за исключением тех из них, которые находятся в текущем выделенном диапазоне. Соответствующий пример показан на рис. 11.12.

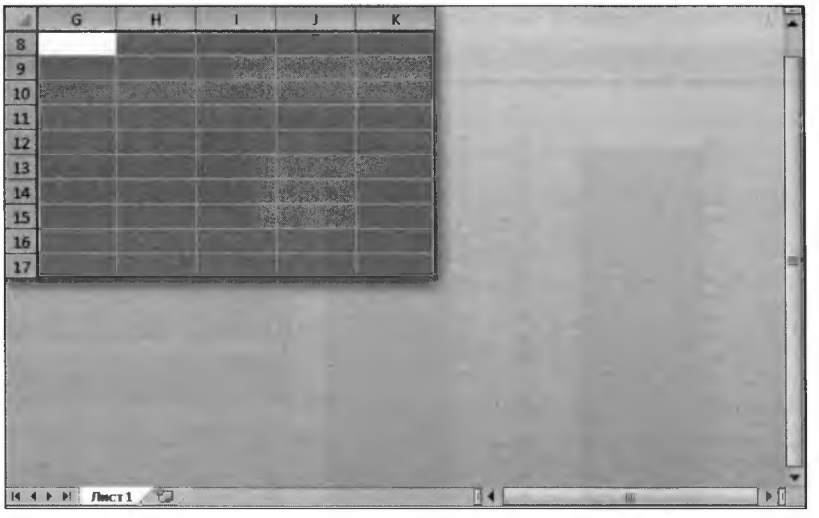


Рис. 11.12. Столбцы и строки скрыты (за исключением диапазона G8:K17)

```
Sub HideRowsAndColumns()
   Dim row1 As Long, row2 As Long
   Dim col1 As Long, col2 As Long

   If TypeName(Selection) <> "Range" Then Exit Sub

'  Если скрыт последний столбец или строка, отобразить все и выйти
   If Rows(Rows.Count).EntireRow.Hidden Or _
      Columns(Columns.Count).EntireColumn.Hidden Then
      Cells.EntireColumn.Hidden = False
      Cells.EntireRow.Hidden = False
      Exit Sub
   End If

   row1 = Selection.Rows(1).Row
   row2 = row1 + Selection.Rows.Count - 1
   col1 = Selection.Columns(1).Columncol2 = col1 + _
      Selection.Columns.Count - 1
   Application.ScreenUpdating = False
   On Error Resume Next
'  Скрыть строки
   Range(Cells(1, 1), Cells(row1 - 1, 1)).EntireRow.Hidden = True
   Range(Cells(row2 + 1, 1), Cells(Rows.Count, _
      1)).EntireRow.Hidden = True
'  Скрыть столбцы
   Range(Cells(1, 1), Cells(1, col1 - 1)).EntireColumn.Hidden = _
      True
      Range(Cells(1, col2 + 1), Cells(1, Columns. _
         Count)).EntireColumn.Hidden = True
End Sub
```

Если выделенный диапазон включает несвязанные диапазоны, первый оператор `Area` применяется в качестве основы для сокрытия строк и столбцов.

Компакт-диск

Рабочая книга, включающая этот пример, находится на прилагаемом компакт-диске в файле `hide rows and columns.xlsm`.

Синхронизация рабочих книг

Если вы работаете с рабочими книгами, состоящими из нескольких листов, то, вероятно, знаете, что Excel не может "синхронизировать" листы в рабочей книге. Другими словами, не существует автоматического способа сделать так, чтобы все листы имели одинаковые выделенные диапазоны и верхние левые ячейки. Макрос VBA, показанный ниже, берет за основу активный рабочий лист и выполняет следующие действия со всеми остальными рабочими листами в книге:

- выделяет тот же диапазон, что и в активном листе;
- задает ту же левую верхнюю ячейку, что и на активном листе.

Ниже приведен листинг данного макроса.

```
Sub SynchSheets()
' Дублирует активный диапазон и верхнюю левую ячейку
' активного листа во всех рабочих листах
   If TypeName(ActiveSheet) <> "Worksheet" Then Exit Sub
   Dim UserSheet As Worksheet, sht As Worksheet
   Dim TopRow As Long, LeftCol As Integer
   Dim UserSel As String

   Application.ScreenUpdating = False
' Сохранение текущего листа
   Set UserSheet = ActiveSheet

' Сохранение информации из активного листа
   TopRow = ActiveWindow.ScrollRow
   LeftCol = ActiveWindow.ScrollColumn
   UserSel = ActiveWindow.RangeSelection.Address

' Просмотр рабочих листов
   For Each sht In ActiveWorkbook.Worksheets
      If sht.Visible Then 'пропуск скрытых листов
         sht.Activate
         Range(UserSel).Select
         ActiveWindow.ScrollRow = TopRow
         ActiveWindow.ScrollColumn = LeftCol
      End If
   Next sht
'  Переход к первоначальной позиции
   UserSheet.Activate
   Application.ScreenUpdating = True
End Sub
```

Компакт-диск

Рабочая книга с рассмотренным выше примером находится на прилагаемом компакт-диске в файле `synchronize sheets.xlsm`.

Методы программирования на VBA

Примеры в этом разделе иллюстрируют часто используемые приемы VBA, которые можно применять в собственных проектах.

Переключение значения булева свойства

Булево свойство — это логическое свойство, принимающее одно из двух значений: `True` (ИСТИНА) или `False` (ЛОЖЬ). Самый простой способ изменить булево свойство — использовать оператор `Not`, как показано в следующем примере, в котором активизируется свойство переноса по словам `WrapText` в выделенном диапазоне ячеек.

```
Sub ToggleWrapText()
'  Управляет переносом слов в выделенных ячейках
   If TypeName(Selection) = "Range" Then
      Selection.WrapText = Not ActiveCell.WrapText
   End If
End Sub
```

Обратите внимание, что за основу взята активная ячейка. Когда диапазон выделен и значения свойств в разных ячейках неодинаковы (например, в одних ячейках шрифт полужирный, а в других — нет), то диапазон считается смешанным, и Excel использует в качестве базового значение свойства активной ячейки. Если, например, активная ячейка имеет полужирный шрифт, то начертание текста в выделенных ячейках после щелчка на кнопке Полужирный панели инструментов будет обычным. Эта простая процедура имитирует поведение элемента интерфейса Excel.

Процедура использует функцию `TypeName`, чтобы проверить, является ли выделенный объект диапазоном. Если это не так, то ничего не происходит.

Оператор `Not` можно использовать для переключения значений многих свойств. Например, для отображения заголовков строк и столбцов в рабочем листе примените следующую команду.

```
ActiveWindow.DisplayHeadings = Not _
   ActiveWindow.DisplayHeadings
```

Для отображения линий сетки на активном листе воспользуйтесь таким кодом.

```
ActiveWindow.DisplayGridlines = Not _
   ActiveWindow.DisplayGridlines
```

Определение количества страниц для печати

Если нужно определить количество страниц для печати, можно использовать команду Excel Предварительный просмотр (Print Preview), а затем подсчитать количество отображающихся на экране страниц. Этот процесс поддается автоматизации с помощью следующей процедуры VBA, которая вычисляет количество страниц для печати на активном листе путем подсчета горизонтальных и вертикальных разрывов страницы.

```
Sub PageCount()
   MsgBox (ActiveSheet.HPageBreaks.Count + 1) * _
      (ActiveSheet.VPageBreaks.Count + 1) & " страниц"
End Sub
```

Представленная ниже процедура VBA циклически просматривает все листы в активной рабочей книге и отображает общее количество страниц для печати (рис. 11.13).

```
Sub ShowPageCount()
   Dim PageCount As Integer
   Dim sht As Worksheet
   PageCount = 0
   For Each sht In Worksheets
      PageCount = PageCount + (sht.HPageBreaks.Count + 1) * _
         (sht.VPageBreaks.Count + 1)
   Next sht
   MsgBox "Всего страниц = " & PageCount
End Sub
```

Компакт-диск

Рабочая книга, включающая рассмотренный пример, находится на прилагаемом компакт-диске в файле `page count.xlsm`.

Отображение даты и времени

Если вы разбираетесь в системе, используемой в Excel для хранения значений дат и времени, то у вас не возникнет проблем при работе со значениями дат и времени в процедурах VBA.

Процедура `DateAndTime` отображает окно сообщения с текущими датой и временем (рис. 11.14). В этом примере в строке заголовка окна сообщения представлено пользовательское сообщение.

Рис. 11.13. С помощью процедуры VBA подсчитывается количество печатных страниц в рабочей книге

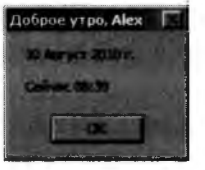


Рис. 11.14. В окне сообщения отображаются дата и время

Процедура использует в качестве аргумента функции `Format` функцию `Date`. В результате строка с датой будет представлена в удобном для восприятия формате. Тот же прием применяется для задания формата времени.

```
Sub DateAndTime()
   Dim TheDate As String, TheTime As String
   Dim Greeting As String
   Dim FullName As String, FirstName As String
   Dim SpaceInName As Long
   TheDate = Format(Date, "Long Date")
   TheTime = Format(Time, "Medium Time")
'  Определение приветствия в зависимости от времени суток
   Select Case Time
      Case Is < TimeValue("12:00"): Greeting = "Доброе утро, "
      Case Is >= TimeValue("17:00"): Greeting = "Добрый вечер, "
      Case Else: Greeting = "Добрый день, "
   End Select
'  Присоединение к приветствию имени пользователя
   FullName = Application.UserName
   SpaceInName = InStr(1, FullName, " ", 1)
```

```
'   Обработка ситуации, когда в имени нет пробела
    If SpaceInName = 0 Then SpaceInName = Len(FullName)
    FirstName = Left(FullName, SpaceInName)
    Greeting = Greeting & FirstName
'   Отображение сообщения
    MsgBox TheDate & vbCrLf & vbCrLf & "Сейчас " & TheTime, vbOKOnly,
    Greeting
End Sub
```

В данном примере использованы именованные форматы ("Long Date" и "Medium Time") для обеспечения работоспособности макроса независимо от региональных настроек компьютера пользователя. Однако вы можете обратиться к другим форматам. Например, чтобы отобразить дату в формате мм/дд/гг, воспользуйтесь следующим оператором:

```
TheDate = Format(Date, "мм/дд/гг")
```

Чтобы отобразить в зависимости от времени суток приветствие в строке заголовка, используется конструкция `Select Case`. Значения времени задаются в VBA так же, как в Excel. Если время меньше 0,5 (полдень), то это утро. Если время больше 0,7083 (5 часов вечера), то это вечер. Все остальное время — это день. Мы выбрали простой способ и использовали функцию VBA `TimeValue`, которая возвращает значение времени из строки.

Следующие операторы определяют имя пользователя, указанное на вкладке **Общие** (General) диалогового окна **Параметры Excel** (Excel Options). Для нахождения первого пробела в имени пользователя использована функция VBA `InStr`. Когда я создавал рассматриваемую процедуру впервые, то не учел, что в имени пользователя пробел может отсутствовать. Поэтому, когда процедура была запущена на компьютере с именем пользователя `Nobody`, программа выдала ошибку, из чего следует, что нельзя предусмотреть все, и даже самые простые процедуры могут дать сбой. (Кстати говоря, если поле для введения имени пользователя не заполнено, то Excel всегда использует значение `User`.) Решение этой проблемы состоит в следующем: присвойте переменной `SpaceInName` длину полного имени пользователя, и функция `Left` извлечет полное имя.

Функция `MsgBox` объединяет дату и время, но использует встроенную константу `vbCrLf` для вставки между ними разрыва строки. `vbOKOnly` — предопределенная константа, возвращающая 0; в результате окно сообщения содержит только кнопку **OK**. Последний аргумент — приветствие `Greeting`, составленное ранее в процедуре.

Компакт-диск

Процедуру `DateAndTime` можно найти на прилагаемом к книге компакт-диске в файле `date and time.xlsm`.

Отображение списка шрифтов

Если вам необходимо познакомиться со списком всех установленных шрифтов, помните, что в Excel нет прямого способа получить эту информацию. Описанная в этом разделе методика основана на том, что в Excel 2010 для обеспечения совместимости поддерживаются свойства и методы объекта `CommandBar`. Эти свойства и методы использовались для работы с панелями инструментов и меню в версиях Excel, предшествующих Excel 2007.

Макрос `ShowInstalledFonts` отображает список установленных шрифтов в столбце A активного рабочего листа. При этом создается временная панель инструментов (объект `CommandBar`), добавляется элемент управления `Font`, а также считываются шрифты из этого элемента управления. Затем временная панель инструментов удаляется.

```
Sub ShowInstalledFonts()
   Dim FontList As CommandBarControl
   Dim TempBar As CommandBar
   Dim i As Long

'  Создание временной панели инструментов CommandBar
   Set TempBar = Application.CommandBars.Add
   Set FontList = TempBar.Controls.Add(ID:=1728)

'  Помещение шрифтов в столбец A
   Range("A:A").ClearContents
   For i = 0 To FontList.ListCount - 1
      Cells(i + 1, 1) = FontList.List(i + 1)
   Next i

'  Удаление временной панели инструментов CommandBar
   TempBar.Delete
End Sub
```

Совет

Дополнительно можно отобразить название каждого шрифта одновременно с его начертанием (рис. 11.15). Для этого в состав цикла `For-Next` добавьте следующий оператор:

```
Cells(i+1,1).Font.Name = FontList.List(i+1)
```

Будьте осторожны, поскольку использование большого количества шрифтов в рабочей книге поглотит ресурсы вашей системы и сможет даже вызвать ее крах.

	A	B	C	D	E	F	G	H
385	Century Schoolbook							
386	Chaparral Pro							
387	CHARLEMAGNE STD							
388	Chiller							
389	Colonna MT							
390	Comic Sans MS							
391	[illegible]							
392	Consolas							
393	Constantia							
394	**Cooper Black**							
395	**Cooper Std Black**							
396	**COPPERPLATE GOTHIC BOLD**							
397	COPPERPLATE GOTHIC LIGHT							
398	Corbel							
399	Courier							
400	Courier New							
401	Curlz MT							
402	DESDEMONA							
403	ECCENTRIC STD							
404	[illegible]							
405	**Elephant**							
406	ENGRAVERS MT							
407	Epsilon							
408	**Eras Bold ITC**							
409	**Eras Demi ITC**							
410	Eras Light ITC							
411	Eras Medium ITC							

Рис. 11.15. Отображение названий и начертаний шрифтов

Компакт-диск

Описанную выше процедуру можно найти на прилагаемом к книге компакт-диске в файле `list fonts.xlsm`.

Сортировка массива

Несмотря на то что в Excel существует встроенная команда сортировки ячеек, в VBA метод сортировки массивов не представлен. Один возможный, но достаточно неудобный способ решения этой задачи — перенести массив в диапазон ячеек на рабочем листе, отсортировать данные с помощью команд Excel и занести результат обратно в массив. Однако если в вашей программе имеет большое значение скорость выполнения операции, то лучше написать на VBA процедуру сортировки.

В данном разделе рассматривается несколько методов сортировки.

- *Сортировка на рабочем листе*. Массив переносится на рабочий лист Excel; диапазон на рабочем листе сортируется и переносится обратно в массив. Единственным аргументом этой процедуры является массив.
- *Пузырьковый метод*. Довольно простой прием сортировки (он использовался в примере сортировки листов в главе 9). Его несложно запрограммировать, однако такой алгоритм сортировки не самый эффективный, особенно для большого количества элементов в массиве.
- *Быстрая сортировка*. Намного более быстрая процедура, чем пузырьковый алгоритм, но, чтобы в ней разобраться, потребуется время. Работает исключительно с типами данных `Integer` и `Long`.
- *Метод пересчета*. Работает очень быстро, однако для его улучшения также потребуются время и определенные усилия. Как и быстрая сортировка, работает с типами данных `Integer` и `Long`.

Компакт-диск

На прилагаемом компакт-диске находится рабочая книга (`sorting demo.xlsm`), демонстрирующая описанные выше методы сортировки. Эту рабочую книгу можно протестировать на массивах разных размеров.

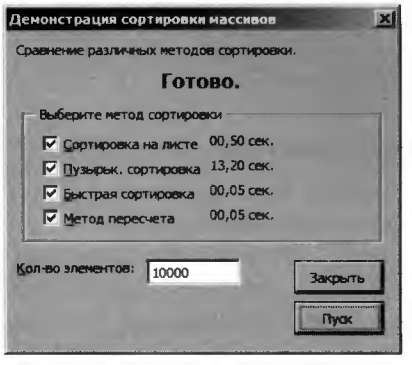


Рис. 11.16. Сравнение времени, необходимого для сортировки массивов разных размеров

На рис. 11.16 показано диалоговое окно для этого проекта. Результаты тестирования получены для семи массивов разных размеров (от 100 до 100 тысяч элементов); элементами массивов выступали произвольные числа (типа `Long`).

В табл. 11.1 представлены результаты теста, выполненного на компьютере редактора книги. Выражение "0,00" означает, что сортировка произошла за время менее 0,01 секунды.

Алгоритм сортировки на рабочем листе поразительно быстрый, принимая во внимание то, что массив переносится на лист, сортируется и затем результаты сортировки переносятся обратно в массив.

Таблица 11.1. Время сортировки (в секундах) для четырех алгоритмов, полученное для массивов с разным количеством элементов

Количество элементов в массиве	Сортировка на рабочем листе	Пузырьковая сортировка VBA	Быстрая сортировка VBA	Сортировка методом пересчета VBA
100	0,03	0,00	0,00	0,03
500	0,05	0,03	0,00	0,03
1000	0,09	0,14	0,02	0,03
5000	0,27	3,28	0,03	0,03
10000	0,52	13,05	0,06	0,05
50000	2,41	317,20	0,31	0,11
100000	4,77	3204,30	0,70	0,19

Алгоритм пузырьковой сортировки относительно быстрый при работе с небольшими массивами, но его лучше не использовать с большими массивами (более 10 тысяч элементов). Алгоритмы быстрой сортировки и сортировки методом пересчета работают очень быстро, но ограничены типами данных `Integer` и `Long`.

Обработка последовательности файлов

Одной из главных причин использования макросов является многократное повторение определенной операции. Пример из этого раздела показывает, как выполнить макрос в нескольких разных файлах, сохраненных на диске. Этот пример, который призван помочь вам написать собственную программу выполнения задачи, запрашивает у пользователя сведения о файле и обрабатывает соответствующие запросу рабочие книги. В рассматриваемом случае обработка предусматривает импорт файла и ввод ряда формул суммирования, описывающих данные в файле.

```
Sub BatchProcess()
   Dim FileSpec As String
   Dim i As Integer
   Dim FileName As String
   Dim FileList() As String
   Dim FoundFiles As Integer
'  Определение пути и сведений о файле
   FileSpec = ThisWorkbook.Path & "\" & "text??.txt"
   FileName = Dir(FileSpec)

'  Найден ли файл?
   If FileName <> "" Then
      FoundFiles = 1
      ReDim Preserve FileList(1 To FoundFiles)
      FileList(FoundFiles) = FileName
   Else
      MsgBox "Не найдены требуемые файлы " & FileSpec
      Exit Sub
   End If

'  Получить другие имена файлов
   Do
      FileName = Dir
      If FileName = "" Then Exit Do
```

```
      FoundFiles = FoundFiles + 1
      ReDim Preserve FileList(1 To FoundFiles)
      FileList(FoundFiles) = FileName & "*"
   Loop

'  Просмотр и обработка файлов
   For i = 1 To FoundFiles
      Call ProcessFiles(FileList(i))
   Next i
End Sub
```

Компакт-диск

Файл рассмотренного выше примера `batch processing.xlsm` находится на прилагаемом к книге компакт-диске. Здесь же можно найти дополнительные файлы `text01.txt`, `text02.txt` и `text03.txt`.

Если нужно импортировать другие текстовые файлы, процедуру придется немного изменить. Соответствующие заданному критерию файлы находятся в массиве `FoundFiles`, а процедура использует для обработки файлов цикл `For-Next`. В цикле обработка выполняется процедурой `ProcessFiles`, показанной ниже. Эта простая процедура использует метод `OpenText` для импорта файла и вставки в него пяти формул. Конечно, вы можете заменить такую процедуру собственной, соответствующей более конкретной задаче.

```
Sub ProcessFiles(FileName As String)
'  Импорт файла
   Workbooks.OpenText FileName:=FileName, _
      Origin:=xlWindows, _
      StartRow:=1, _
      DataType:=xlFixedWidth, _
      FieldInfo:= _
      Array(Array(0, 1), Array(3, 1), Array(12, 1))
'  Ввод формул суммирования
   Range("D1").Value = "A"
   Range("D2").Value = "B"
   Range("D3").Value = "C"
   Range("E1:E3").Formula = "=COUNTIF(B:B,D1)"
   Range("F1:F3").Formula = "=SUMIF(B:B,D1,C:C)"
End Sub
```

Перекрестная ссылка

Дополнительные сведения о работе с файлами с помощью VBA можно найти в главе 27.

Полезные функции для программ VBA

В данном разделе представлены некоторые "практичные" функции, которые будут использоваться в ваших приложениях либо помогут в создании аналогичных функций. Эти функции наиболее полезны, когда вызываются из другой процедуры VBA. Следовательно, они объявляются с ключевым словом `Private` и не отображаются в диалоговом окне Excel Мастер функций (Insert Function).

Компакт-диск

Примеры, описанные в этом разделе, можно найти на прилагаемом к книге компакт-диске в файле VBA `utility functions.xlsm`.

Функция `FileExists`

Данная функция получает один аргумент (путь и имя файла) и возвращает ИСТИНА, если файл существует.

```
Private Function FileExists(fname) As Boolean
' Возвращает ИСТИНА, если файл существует
  FileExists = (Dir(fname) <> "")
End Function
```

Функция `FileNameOnly`

Функция получает один аргумент (путь и имя файла) и возвращает только имя файла. Другими словами, функция "обрезает" путь.

```
Private Function FileNameOnly(pname) As String
' Возвращает имя файла из строки путь/имя файла
  Dim temp As Variant
  length = Len(pname)
  temp = Split(pname, Application.PathSeparator)
  FileNameOnly = temp(UBound(temp))
End Function
```

Функция использует функцию VBA `Split`, которая принимает строку (вместе с символами-разделителями) и возвращает массив типа `variant`, содержащий элементы, которые находятся между символами-разделителями. В рассматриваемом случае переменной `temp` присваивается массив, содержащий текстовые строки между `Application.PathSeparater` (обычно в качестве разделителя используется обратная косая черта). Дополнительные примеры использования функции `Split` будут рассмотрены далее.

Если в качестве аргумента указать `c:\excel files\2010\backup\budget.xlsx`, функция возвратит строку `budget.xlsx`.

Функция `FileNameOnly` обрабатывает любой путь и имя файла (даже если файла не существует). Если файл существует, лучше воспользоваться следующей более простой функцией.

```
Private Function FileNameOnly2(pname) As String
  FileNameOnly2 = Dir(pname)
End Function
```

Функция `PathExists`

Функция получает один аргумент (путь) и возвращает ИСТИНА, если путь существует.

```
Private Function PathExists(pname) As Boolean
' Возвращает ИСТИНА, если путь существует
  If Dir(pname, vbDirectory) = "" Then
     PathExists = False
  Else
     PathExists = (GetAttr(pname) And vbDirectory) = vbDirectory
  End If
End Function
```

Функция RangeNameExists

Функция получает один аргумент (название диапазона) и возвращает ИСТИНА, если в активной рабочей книге существует указанное название диапазона.

```
Private Function RangeNameExists(nname) As Boolean
'  Возвращает ИСТИНА, если имя диапазона существует
   Dim n As Name
   RangeNameExists = False
   For Each n In ActiveWorkbook.Names
      If UCase(n.Name) = UCase(nname) Then
         RangeNameExists = True
         Exit Function
      End If
   Next n
End Function
```

Ниже представлен еще один вариант этой функции. Она создает объектную переменную, использующую имя. Если имени не существует, генерируется ошибка.

```
Private Function RangeNameExists2(nname) As Boolean
'  Возвращает ИСТИНА, если имя диапазона существует
   Dim n As Range
   On Error Resume Next
   Set n = Range(nname)
   If Err.Number = 0 Then RangeNameExists2 = True _
      Else RangeNameExists2 = False
End Function
```

Функция SheetExists

Функция получает один аргумент (название рабочего листа) и возвращает ИСТИНА, если данный рабочий лист существует в активной рабочей книге.

```
Private Function SheetExists(sname) As Boolean
'  Возвращает ИСТИНА, если рабочий лист существует в активной
'  рабочей книге
   Dim x As Object
   On Error Resume Next
   Set x = ActiveWorkbook.Sheets(sname)
   If Err.Number = 0 Then SheetExists = True _
      Else SheetExists = False
End Function
```

Функция WorkbookIsOpen

Функция получает один аргумент (название рабочей книги) и возвращает ИСТИНА, если данная рабочая книга открыта.

```
Private Function WorkbookIsOpen(wbname) As Boolean
'  Возвращает ИСТИНА, если рабочая книга открыта
   Dim x As Workbook
   On Error Resume Next
   Set x = Workbooks(wbname)
   If Err.Number = 0 Then WorkbookIsOpen = True _
      Else WorkbookIsOpen = False
End Function
```

Проверка принадлежности к коллекции

Следующая функция представляет собой образец "групповой" функции, с помощью которой можно определить, является ли объект членом коллекции.

```
Private Function IsInCollection(Coln As Object, _
   Item As String) As Boolean
   Dim Obj As Object
   On Error Resume Next
   Set Obj = Coln(Item)
   IsInCollection = Not Obj Is Nothing
End Function
```

Эта функция имеет два аргумента: коллекцию (объект) и элемент (строка), который может быть либо не быть членом коллекции. Функция будет создавать объектную переменную, представляющую элемент коллекции. Если попытка увенчается успехом, функция возвратит `True`; иначе — `False`.

Функцию `IsInCollection` можно использовать вместо трех других функций, перечисленных в этой главе: `RangeNameExists`, `SheetExists` и `WorkbookIsOpen`. Чтобы определить, содержится ли в активной рабочей книге диапазон `Data`, вызовите функцию `IsInCollection` с помощью следующего оператора:

```
MsgBox IsInCollection(ActiveWorkbook.Names, "Data")
```

Для того чтобы определить, открыта ли рабочая книга с названием `Budget`, используйте следующий оператор:

```
MsgBox IsInCollection(Workbooks, "budget.xlsx")
```

Чтобы узнать, содержит ли активная рабочая книга рабочий лист `Лист1`, используйте следующий оператор:

```
MsgBox IsInCollection(ActiveWorkbook.Worksheets, "Лист1")
```

Получение значения из закрытой рабочей книги

В VBA не существует метода получения значения из закрытого файла рабочей книги. Однако вы можете воспользоваться возможностью управления ссылками на файлы, которая предоставляется в Excel. В настоящем разделе описана функция VBA (`GetValue`, показанная ниже), которая получает значение из закрытой книги. Эта задача выполняется в результате вызова *макроса XLM*, который появился в "доисторических" версиях Excel (до версии 5), но поддерживается до сих пор.

```
Private Function GetValue(path, file, sheet, ref)
' Выборка значения из закрытой рабочей книги
   Dim arg As String
' Проверка существования файла
   If Right(path, 1) <> "\" Then path = path & "\"
   If Dir(path & file) = "" Then
      GetValue = "Файл не найден "
      Exit Function
   End If
' Создание аргумента
   arg = "'" & path & "[" & file & "]" & sheet & "'!" & _
      Range(ref).Range("A1").Address(, , xlR1C1)
' Выполнение макроса XLM
   GetValue = ExecuteExcel4Macro(arg)
End Function
```

Функция GetValue принимает четыре аргумента:

- path — путь к закрытому файлу (например, "d:\files");
- file — название рабочей книги (например, "budget.xls");
- sheet — название рабочего листа (например, "Лист1");
- ref — ссылка на ячейку (например, "C4").

Следующая процедура демонстрирует, как используется функция GetValue. В этой процедуре отображается значение ячейки A1 листа Лист1 файла 2010Budget.xlsx (папка XLFiles\Budget на диске C:).

```
Sub TestGetValue()
   Dim p As String, f As String
   Dim s As String, a As String
   p = "c:\XLFiles\Budget"
   f = "2010budget.xlsx"
   s = "Лист1"
   a = "A1"
   MsgBox GetValue(p, f, s, a)
End Sub
```

Ниже приведен еще один пример. Эта процедура считывает 1200 значений (100 строк и 12 столбцов) из закрытого файла и помещает эти значения на активный рабочий лист.

```
Sub TestGetValue2()
   Dim p As String, f As String
   Dim s As String, a As String
   Dim r As Long, c As Long
   p = "c:\XLFiles\Budget"
   f = "2010Budget.xlsx"
   s = "Лист1"
   Application.ScreenUpdating = False
   For r = 1 To 100
      For c = 1 To 12
         a = Cells(r, c).Address
         Cells(r, c) = GetValue(p, f, s, a)
      Next c
   Next r
End Sub
```

Примечание

Функция GetValue не работает, если ее использовать в формуле рабочего листа. Эту функцию вообще не следует использовать, поскольку для получения значения из закрытого файла можно просто создать формулу со ссылкой.

Компакт-диск

Описанный в этом разделе пример можно найти на прилагаемом к книге компакт-диске в файле value from a closed workbook.xlsm. При работе с закрытым файлом в примере используется файл myworkbook.xlsx.

Полезные функции в формулах Excel

Примеры, приведенные в этом разделе, представляют пользовательские функции, которые можно применять в формулах рабочего листа. Помните, что эти процедуры функций необходимо определить в модуле VBA, а не в модуле кода соответствующей рабочей книги Эта книга (ThisWorkbook), листа или пользовательской формы (UserForm).

Компакт-диск

Примеры из этого раздела можно найти на прилагаемом к книге компакт-диске в файле `worksheet functions.xlsm`.

Получение информации о форматировании ячейки

В данном разделе содержится ряд пользовательских функций, возвращающих информацию о форматировании ячейки. Такие функции используются при сортировке данных на основе форматирования (например, в случае, когда ячейки, выделенные полужирным шрифтом, должны располагаться рядом).

Предупреждение

Вскоре вы сможете убедиться в том, что эти специальные функции не всегда обновляются автоматически — изменение форматирования не приводит к пересчету формул Excel. Чтобы вызвать глобальный пересчет формул (и обновить все пользовательские функции), нажмите клавиши <Ctrl+Alt+F9>.

В функцию можно также добавить следующий оператор:

```
Application.Volatile
```

При наличии этого оператора пересчет функции производится после нажатия клавиши <F9>.

Следующая функция возвращает ИСТИНА, если аргумент, состоящий из одной ячейки, выделен полужирным шрифтом. Если диапазон передается в качестве аргумента, функция использует его верхнюю левую ячейку.

```
Function IsBold(cell) As Boolean
' Возвращает ИСТИНА, если ячейка выделена полужирным
  IsBold = cell.Range("A1").Font.Bold
End Function
```

Помните о том, что эти функции работают только с явно заданным форматированием (не могут применяться с условным форматированием). В Excel 2010 появился новый объект под названием `DisplayFormat`. Он учитывает наличие условного форматирования. Ниже показан код функции `IsBold`, который может обрабатывать полужирный формат, являющийся результатом условного форматирования.

```
Function IsBold(cell) As Boolean
' Возвращает ИСТИНА даже в случае условного форматирования
  IsBold = cell.Range("A1").DisplayFormat.Font.Bold
End Function
```

Следующая функция возвращает ИСТИНА, если используемая в качестве аргумента одна ячейка выделена курсивом.

```
Function IsItalic(cell) As Boolean
' Возвращает ИСТИНА, если ячейка выделена курсивом
  IsItalic = cell.Range("A1").Font.Italic
End Function
```

Обе предыдущие функции возвращают ошибку, если ячейка имеет смешанное форматирование (например, полужирным шрифтом отображены только отдельные символы). Функция, приведенная ниже, возвращает ИСТИНА только тогда, когда все символы в ячейке выделены полужирным шрифтом.

```
Function AllBold(cell) As Boolean
' Возвращает ИСТИНА, если все символы в ячейке
```

```
'  выделены полужирным
   If IsNull(cell.Font.Bold) Then
      AllBold = False
   Else
      AllBold = cell.Font.Bold
   End If
End Function
```

Функцию `AllBold` можно упростить.

```
Function AllBold (cell) As Boolean
'  Возвращает ИСТИНА, если все символы в ячейке
'  выделены полужирным
   AllBold = Not IsNull(cell.Font.Bold)
End Function
```

Функция `FillColor`, представленная далее, возвращает целое число, соответствующее индексу цвета фона ячейки (цвета заливки ячейки). Если ячейка не имеет заливки, то функция возвращает значение `4142`.

Эта функция не может использоваться для определения цветов заливки таблиц (которые создаются с помощью команды **Вставка⇨Таблицы⇨Таблица** (Insert⇨Tables⇨Table)) или сводных таблиц. В подобных случаях воспользуйтесь объектом `DisplayFormat`, как описывалось ранее.

```
Function FillColor(cell) As Integer
'  Возвращает целое число, соответствующее
'  цвету ячейки
   FillColor = cell.Range("A1").Interior.ColorIndex
End Function
```

Беседа с рабочим листом

Функция `SayIt` применяет синтезатор речи Excel для "проговаривания" аргументов, в качестве которых используется текст или ссылка на ячейку.

```
Function SayIt(txt)
   Application.Speech.Speak (txt)
   SayIt = txt
End Function
```

Эта функция носит развлекательный характер, но может использоваться и в серьезных целях. Например, ее можно включить в следующую формулу:

```
=IF(SUM(A:A)>25000,SayIt("Цель достигнута "))
```

Если сумма значений в столбце A превышает 25000, вы услышите синтезированный голос, сообщающий о том, что цель достигнута. Метод `Speak` можно также включить в конец длинной процедуры — спустя некоторое время компьютер известит вас о том, что выполнение процедуры завершено.

Отображение даты сохранения файла или вывода файла на печать

Рабочая книга Excel содержит несколько встроенных свойств документа, к которым можно получить доступ с помощью свойства `BuiltinDocumentProperties` объекта `Workbook`. Следующая функция возвращает дату и время последнего сохранения рабочей книги.

```
Function LastSaved()
   Application.Volatile
   LastSaved = ThisWorkbook. _
```

```
    BuiltinDocumentProperties("Время последнего сохранения ")
End Function
```

Значения даты и времени, возвращаемые этой функцией, совпадают со значениями даты и времени, которые отображаются в разделе **Связанные даты** (Related Dates) окна представления Backstage. Это окно отображается после выбора команды **Файл⇨ Сведения** (File⇨Info). Обратите внимание, что на значения даты и времени оказывает влияние свойство `AutoSave`. Поэтому "время последнего сохранения" необязательно имеет отношение ко времени сохранения файла пользователем.

Показанная ниже функция напоминает предыдущую, но возвращает дату и время последнего вывода рабочей книги на печать или предварительного просмотра рабочей книги. Если рабочая книга никогда не печаталась и не просматривалась, функция возвращает ошибку `#ЗНАЧ`.

```
Function LastPrinted()
   Application.Volatile
   LastPrinted = ThisWorkbook. _
      BuiltinDocumentProperties("Дата последнего вывода на печать")
End Function
```

При использовании этих функций в формуле необходимо вызвать пересчет формул (клавиша <F9>), чтобы получить текущие значения данных свойств.

Примечание

Существуют еще несколько встроенных свойств, но Excel их не использует. Например, при попытке получить доступ к свойству `Number of Bytes`, указывающему размер файла, будет генерироваться ошибка. Для получения списка встроенных свойств обратитесь к справочной системе.

Приведенные выше функции `lastSave` и `lastPrinted` предназначались для сохранения в той рабочей книге, в которой они используются. В отдельных случаях требуется сохранить функцию в книге, отличной от той (например, `personal.xlsb`), в которой она используется, или в надстройке. Поскольку все функции ссылаются на книгу `Эта книга` (ThisWorkbook), то выполняться корректно они не будут. Следуйте приведенным ниже инструкциям для создания универсальных функций. В данных процедурах используется метод `Application.Caller`, который возвращает объект `Range`. Этот объект указывает на ячейку, из которой вызывается функция. Оператор `Parent.Parent` возвращает рабочую книгу (родитель родителя объекта `Range` — объект `Workbook`). Взаимоотношения родительских и дочерних объектов детально рассмотрены далее.

```
Function LastSaved2()
   Application.Volatile
   LastSaved2 = Application.Caller.Parent.Parent. _
      BuiltinDocumentProperties("Время последнего сохранения")
End Function
```

Основы иерархии объектов

Объектная модель Excel представляет собой определенную структуру: одни объекты содержатся в других объектах. На вершине этой иерархии находится объект `Application`. Excel содержит другие объекты, в которые, в свою очередь, вложены более низкоуровневые объекты и т.д. Следующая иерархия показывает, как в этой структуре представлен объект `Range`.

```
Объект Application
   Объект Workbook
      Объект Worksheet
         Объект Range
```

На языке объектно-ориентированного программирования родителем объекта Range (Диапазон) является объект Worksheet (Рабочий лист), в котором он содержится. Родителем объекта Worksheet является объект Workbook (Рабочая книга), содержащий этот рабочий лист, а родителем объекта Workbook выступает объект Application (приложение, т.е. Excel).

Как можно применить эту информацию на практике? Проанализируем функцию VBA SheetName, показанную ниже. Данная функция получает один аргумент (диапазон) и возвращает имя рабочего листа, который содержит указанный диапазон. При этом используется свойство Parent объекта Range. Свойство Parent возвращает объект, а именно — объект, содержащий объект Range.

```
Function SheetName(ref) As String
   SheetName = ref.Parent.Name
End Function
```

Функция WorkbookName возвращает название рабочей книги для конкретной ячейки. Обратите внимание, что эта функция использует свойство Parent дважды. Первое свойство Parent возвращает объект Worksheet, а второе свойство Parent возвращает объект Workbook.

```
Function WorkbookName(ref) As String
   WorkbookName = ref.Parent.Parent.Name
End Function
```

Функция AppName, показанная далее, переносит это упражнение на следующий логический уровень, обращаясь к свойству Parent трижды. Такая функция возвращает имя объекта Application для заданной ячейки. Указанная функция всегда будет возвращать значение Microsoft Excel.

```
Function AppName(ref) As String
   AppName = ref.Parent.Parent.Parent.Name
End Function
```

Подсчет количества ячеек между двумя значениями

Функция CountBetween возвращает количество значений в диапазоне (первый аргумент), которые попадают в область, заданную вторым и третьим аргументами.

```
Function CountBetween(InRange, num1, num2) As Long
'  Подсчитывает количество значений между num1 и num2
   With Application.WorksheetFunction
      If num1 <= num2 Then
         CountBetween = .CountIf(InRange, ">=" & num1) - _
            .CountIf(InRange, ">" & num2)
      Else
         CountBetween = .CountIf(InRange, ">=" & num2) - _
            .CountIf(InRange, ">" & num1)
      End If
   End With
End Function
```

Обратите внимание, что эта функция вызывает функцию Excel СЧЁТЕСЛИ (COUNTIF). По сути, функция CountBetween является "оболочкой", которая может упростить формулы.

Примечание

Обратите внимание, что функция СЧЁТЕСЛИ появилась в Excel 2007. Поэтому приведенный выше код не будет работать с предыдущими версиями Excel.

Ниже приведен пример формулы, использующей функцию CountBetween. Формула возвращает количество ячеек в диапазоне A1:A100, которое больше или равно 10 и меньше или равно 20.

```
=CountBetween(A1:A100,10,20)
```

Применяйте эту функцию VBA, чтобы не вводить следующую длинную формулу:

```
=СЧЁТЕСЛИ(A1:A100,">=10")-СЧЁТЕСЛИ(A1:A100,">20")
```

Определение последней непустой ячейки в столбце или в строке

В этом разделе будут рассмотрены две полезные функции: LastInColumn, которая возвращает содержимое последней непустой ячейки в столбце, и LastInRow, которая возвращает содержимое последней непустой ячейки в строке. В качестве единственного аргумента эти функции используют диапазон. Причем в качестве диапазона может применяться как весь столбец (функция LastInColumn), так и вся строка (функция LastInRow). Если же в качестве аргумента не используется вся строка либо весь столбец, задействуется строка или столбец, в котором находится верхняя левая ячейка диапазона. Например, следующая формула возвращает последнее значение в столбце B:

```
=LastInColumn(B5)
```

Следующая формула возвращает последнее значение в строке 7:

```
=LastInRow(C7:D9)
```

Ниже представлен код функции LastInColumn.

```
Function LastInColumn(rng As Range)
' Возвращает содержимое последней непустой ячейки столбца
  Dim LastCell As Range
  Application.Volatile
  With rng.Parent
     With .Cells(.Rows.Count, rng.Column)
        If Not IsEmpty(.Value) Then
           LastInColumn = .Value
        ElseIf IsEmpty(.End(xlUp)) Then
           LastInColumn = ""
        Else
           LastInColumn = .End(xlUp).Value
        End If
     End With
  End With
End Function
```

Эта функция довольно сложная, поэтому ниже приведено несколько замечаний, которые помогут вам в ней разобраться.

- Оператор Application.Volatile вызывает выполнение функции всякий раз, когда пересчитываются формулы на рабочем листе.
- Оператор Rows.Count возвращает количество строк на рабочем листе. Используется именно он, а не жестко заданное значение, из соображений совместимости (новые версии Excel могут включать большее количество строк на рабочем листе).

- Ссылка `rng.Column` возвращает номер столбца левой верхней ячейки в аргументе `rng`.
- Благодаря ссылке `rng.Parent` функция работает корректно, даже если аргумент `rng` ссылается на другой лист или рабочую книгу.
- Метод `End` (с аргументом `xlUp`) эквивалентен переходу к последней ячейке столбца и нажатию <End> и <↑>.
- Функция `IsEmpty` проверяет, пуста ли ячейка. Если ячейка пуста, функция возвращает пустую строку. Без этого оператора пустой ячейке соответствовал бы результат 0.

Ниже представлен код функции `LastInRow`. Она подобна функции `LastInColumn`.

```
Function LastInRow(rng As Range)
'  Возвращает содержимое последней непустой ячейки в строке
   Application.Volatile
   With rng.Parent
      With .Cells(rng.Row, .Columns.Count)
         If Not IsEmpty(.Value) Then
            LastInRow = .Value
         ElseIf IsEmpty(.End(xlToLeft)) Then
            LastInRow = ""
         Else
            LastInRow = .End(xlToLeft).Value
         End If
      End With
   End With
End Function
```

Соответствует ли строка шаблону

Функция `IsLike` довольно проста (и очень полезна). Она возвращает значение ИСТИНА, если строка соответствует заданному шаблону.

Код функции `IsLike` показан далее. Она представляет собой "оболочку", позволяющую использовать в формулах мощный оператор VBA `Like`.

```
Function IsLike(text As String, pattern As String) As Boolean
'  Возвращает ИСТИНА, если первый оператор подобен второму
   IsLike = text Like pattern
End Function
```

Функция `IsLike` принимает два аргумента:

- `text` — текстовая строка или ссылка на ячейку, содержащую текстовую строку;
- `pattern` — строка, содержащая групповые символы согласно следующему списку.

Символ в шаблоне	Соответствие в тексте
`?`	Любой отдельный символ
`*`	Нуль и более символов
`#`	Любая отдельная цифра (`0-9`)
`[`*список_символов*`]`	Любой отдельный символ из перечня *список_символов*
`[!`*список_символов*`]`	Любой отдельный символ, не принадлежащий перечню *список_символов*

Представленная ниже формула возвращает ИСТИНА, так как * соответствует любому количеству символов. Она возвращает ИСТИНА, если первый аргумент — любой текст, начинающийся с g.

```
=IsLike("guitar","g*")
```

Следующая формула возвращает ИСТИНА, так как ? соответствует любому отдельному символу. Если бы первым аргументом функции был "Unit12", то функция возвращала бы ЛОЖЬ.

```
=IsLike("Unit1","Unit?")
```

Показанная далее формула возвращает ИСТИНА, так как первый аргумент является одним из символов списка во втором аргументе.

```
=IsLike("a","[aeiou]")
```

Следующая формула возвращает ИСТИНА, если ячейка A1 содержит один из символов: a, e, i, o, u, A, E, I, O, U. При использовании функции ПРОПИСН (UPPER) в аргументе функция становится нечувствительной к регистру.

```
=IsLike(ПРОПИСН(A1), ПРОПИСН("[aeiou]"))
```

Представленная далее формула возвращает ИСТИНА, если в ячейке A1 находится значение, начинающееся с 1 и состоящее ровно из трех цифр (т.е. любое целое число от 100 до 199).

```
=IsLike(A1,"1##")
```

Возвращение из строки n-го элемента

ExtractElement — специальная функция рабочего листа (ее можно также вызвать из процедуры VBA), которая помогает извлечь элемент из текстовой строки. Например, если ячейка содержит следующий текст, вы можете использовать функцию ExtractElement для извлечения любых подстрок между дефисами.

```
123-456-789-0133-8844
```

Представленная далее формула, например, возвращает 0133, т.е. четвертый элемент в строке. Дефис (-) используется в строке как разделитель.

```
=ExtractElement("123-456-789-0133-8844",4,"-")
```

Функция ExtractElement принимает три аргумента:

- txt — текстовая строка, из которой извлекается подстрока (это может быть символьная строка или ссылка на ячейку);
- n — целое число, представляющее номер извлекаемого элемента;
- Separator — отдельный символ, используемый как разделитель.

Примечание

Если в качестве символа-разделителя задать пробел, то несколько пробелов будут рассматриваться как один, что не всегда соответствует требованиям. Если n превышает количество элементов в строке, функция возвращает пустую строку.

Ниже приводится код VBA для функции ExtractElement.

```
Function ExtractElement(Txt, n, Separator) As String
' Возвращает n-й элемент текстовой строки, где
```

```
'   элементы разделены указанным символом-разделителем
    Dim AllElements As Variant
    AllElements = Split(Txt, Separator)
    ExtractElement = AllElements(n - 1)

End Function
```

В этой процедуре используется VBA-функция `Split`, возвращающая массив констант, из которого состоит текстовая строка. Массив начинается с нулевого элемента (а не с первого), поэтому текущий элемент имеет индекс `n-1`.

Преобразование чисел в текст

Функция `SPELLDOLLARS` возвращает текст, в который преобразуется исходное число (подобная операция часто выполняется для проверки и подтверждения указанной суммы в платежных документах). Например, следующая формула возвращает строку `двадцать три и 45/100 доллара`:

```
=SPELLDOLLARS(123,45)
```

На рис. 11.17 показаны некоторые дополнительные примеры использования функции `SPELLDOLLARS`. Формулы, использующие эту функцию, находятся в столбце C. Например, формула в столбце C1 имеет следующий вид:

```
=SPELLDOLLARS(A1)
```

Обратите внимание, что отрицательные числа заключаются в круглые скобки.

	A	B	
1	32		тридцать-два и 00/100 Dollars
2	37,56		тридцать-семь и 56/100 Dollars
3	-32		(тридцать-два и 00/100 Dollars)
4	-26,44		(двадцать-шесть и 44/100 Dollars)
5	-4		(четыре и 00/100 Dollars)
6	1,87341		один и 87/100 Dollars
7	1,56		один и 56/100 Dollars
8	1		один и 00/100 Dollars
9	6,56		шесть и 56/100 Dollars
10	12,12		двенадцать и 12/100 Dollars
11	1000000		один миллион и 00/100 Dollars
12	1000000000		один миллиард и 00/100 Dollars
13			
14			
15			
16			

Рис. 11.17. Примеры использования функции `SPELLDOLLARS`

Компакт-диск

Функция `SPELLDOLLARS` слишком громоздкая, чтобы приводить здесь полностью ее код. Обратитесь к файлу `worksheet function.xlsm` на прилагаемом к книге компакт-диске.

Универсальная функция

В этом примере рассматривается прием, позволяющий сделать так, чтобы одна функция рабочего листа работала как несколько функций. Например, ниже показан код VBA для специальной функции `StatFunction`. Эта функция имеет два аргумента: диапазон

(rng) и операция (op). В зависимости от значения аргумента op функция возвращает значение, вычисленное с помощью одной из следующих функций Excel: СРЗНАЧ (AVERAGE), СЧЁТ (COUNT), МАКС (MAX), МЕДИАНА (MEDIAN), МИН (MIN), МОД (MODE), СТАНДОТКЛОН (STDEV), СУММ (SUM) или ДИСП (VAR).

Например, можно использовать эту функцию на рабочем листе следующим образом:

```
=StatFunction(B1:B24,A24)
```

Результат выполнения формулы зависит от содержимого ячейки A24, которое представлено строкой: СРЗНАЧ (AVERAGE), СЧЁТ (COUNT), МАКС (MAX) и т.д. Можете применить этот прием в других типах функций.

```
Function StatFunction(rng, op)
  Select Case UCase(op)
     Case "СУММ"
        StatFunction = WorksheetFunction.Sum(rng)
     Case "СРЗНАЧ"
        StatFunction = WorksheetFunction.Average(rng)
     Case "МЕДИАНА"
        StatFunction = WorksheetFunction.Median(rng)
     Case "МОД"
        StatFunction = WorksheetFunction.Mode(rng)
     Case "СЧЁТ"
        StatFunction = WorksheetFunction.Count(rng)
     Case "МАКС"
        StatFunction = WorksheetFunction.Max(rng)
     Case "МИН"
        StatFunction = WorksheetFunction.Min(rng)
     Case "ДИСП"
        StatFunction = WorksheetFunction.Var(rng)
     Case "СТАНДОТКЛОН"
        StatFunction = WorksheetFunction.StDev(rng)
     Case Else
        StatFunction = CVErr(xlErrNA)
  End Select
End Function
```

Функция SheetOffset

В Excel ограничена поддержка "трехмерных рабочих книг". Например, чтобы сослаться на другой рабочий лист в книге, включите в формулу имя рабочего листа. Данная проблема будет оставаться незначительной до тех пор, пока вы не попытаетесь скопировать формулу из одного листа в другой. Скопированные формулы продолжают ссылаться на первоначальное имя рабочего листа, и ссылки на листы не изменяются, как это происходит в реальной трехмерной рабочей книге.

Пример, рассмотренный в этом разделе, представляет функцию VBA SheetOffset, которая обеспечивает установку относительных ссылок на рабочие листы. Например, можно сослаться на ячейку A1 предыдущего рабочего листа с помощью такой формулы:

```
=SheetOffset(-1,A1)
```

Первый аргумент представляет лист и может быть положительным, отрицательным или нулевым. Второй аргумент должен быть ссылкой на одну ячейку. Можете скопировать эту формулу в другие листы, и в скопированных формулах будет использована относительная ссылка.

Ниже приведен код VBA функции SheetOffset.

```
Function SheetOffset(Offset As Long, Optional Cell As Variant)
'  Возвращение содержимого ячейки по ссылке на нее
   Dim WksIndex As Long, WksNum As Long
   Dim wks As Worksheet
   Application.Volatile
   If IsMissing(Cell) Then Set Cell = Application.Caller
   WksNum = 1
   For Each wks In Application.Caller.Parent.Parent.Worksheets
      If Application.Caller.Parent.Name = wks.Name Then
         SheetOffset = Worksheets(WksNum + _
            Offset).Range(Cell(1).Address)
         Exit Function
      Else
         WksNum = WksNum + 1
      End If
   Next wks
End Function
```

Возвращение максимального значения всех рабочих листов

Если необходимо определить максимальное значение в ячейке B1 в нескольких рабочих листах, используется следующая формула:

```
=МАКС(Лист1:Лист4!B1)
```

Эта формула возвращает максимальное значение ячейки B1 для листов Лист1, Лист4 и всех листов между ними.

Что же произойдет, если добавить после листа Лист4 новый лист (Лист5)? Формула не будет автоматически изменена, поэтому ее необходимо отредактировать, чтобы включить ссылку на новый лист.

```
=MAX(Лист1:Лист5!B1)
```

Функция `MaxAllSheets`, показанная ниже, получает аргумент (одна ячейка) и возвращает максимальное значение в этой ячейке во всех рабочих листах данной книги. Например, следующая формула возвращает максимальное значение в ячейке B1 для всех листов книги.

```
=MaxAllSheets(B1)
```

При добавлении нового листа редактировать формулу необязательно.

```
Function MaxAllSheets(cell)
   Dim MaxVal As Double
   Dim Addr As String
   Dim Wksht As Object
   Application.Volatile
   Addr = cell.Range("A1").Address
   MaxVal = -9.9E+307
   For Each Wksht In cell.Parent.Parent.Worksheets
      If Wksht.Name = cell.Parent.Name And _
         Addr = Application.Caller.Address Then
         ' избежание циклической ссылки
      Else
         If IsNumeric(Wksht.Range(Addr)) Then
            If Wksht.Range(Addr) > MaxVal Then _
               MaxVal = Wksht.Range(Addr).Value
            End If
```

```
        End If
    Next Wksht
    If MaxVal = -9.9E+307 Then MaxVal = 0
    MaxAllSheets = MaxVal
End Function
```

Оператор `For Each` использует для доступа к рабочей книге следующее выражение:

```
cell.Parent.Parent.Worksheets
```

"Родителем" ячейки является рабочий лист, "родителем" рабочего листа — рабочая книга. Следовательно, цикл `For Each` проходит по всем рабочим листам в книге. Первый оператор `If` внутри цикла проверяет, содержит ли ячейка, которая проверяется в данный момент, функцию. Если содержит, то ячейка игнорируется во избежание циклической ссылки.

Примечание

Функцию можно легко изменить, чтобы приспособить ее к выполнению вычислений в нескольких рабочих листах — к определению минимального и среднего значений, а также суммы.

Возвращение массива случайных целых чисел без повторов

Функция `RandomIntegers`, представленная в этом разделе, возвращает массив целых чисел без повторов. Она предназначена для применения в формуле массива в нескольких ячейках.

```
{=RandomIntegers()}
```

Данная формула введена в весь диапазон с помощью комбинации клавиш <Ctrl+Shift+Enter>. Она возвращает массив целых чисел без повторов, упорядоченных произвольным образом. Так как формулу содержат 50 ячеек, целые числа указываются в диапазоне от 1 до 50.

Ниже представлен код функции `RandomIntegers`.

```
Function RandomIntegers()
    Dim FuncRange As Range
    Dim V() As Variant, ValArray() As Variant
    Dim CellCount As Double
    Dim i As Integer, j As Integer
    Dim r As Integer, c As Integer
    Dim Temp1 As Variant, Temp2 As Variant
    Dim RCount As Integer, CCount As Integer

'   Создание объекта Range
    Set FuncRange = Application.Caller
'   Возвращение ошибки, если диапазон FuncRange слишком большой
    CellCount = FuncRange.Count
    If CellCount > 1000 Then
       RandomIntegers = CVErr(xlErrNA)
       Exit Function
    End If

'   Присваивание значений переменным
    RCount = FuncRange.Rows.Count
    CCount = FuncRange.Columns.Count
    ReDim V(1 To RCount, 1 To CCount)
```

```
   ReDim ValArray(1 To 2, 1 To CellCount)
'  Заполнение массива случайными номерами
'  и последовательными целыми числами
   For i = 1 To CellCount
      ValArray(1, i) = Rnd
      ValArray(2, i) = i
   Next i
'  Сортировка массива Sort ValArray по произвольным числам
   For i = 1 To CellCount
      For j = i + 1 To CellCount
         If ValArray(1, i) > ValArray(1, j) Then
            Temp1 = ValArray(1, j)
            Temp2 = ValArray(2, j)
            ValArray(1, j) = ValArray(1, i)
            ValArray(2, j) = ValArray(2, i)
            ValArray(1, i) = Temp1
            ValArray(2, i) = Temp2
         End If
      Next j
   Next i

'  Занесение произвольных значений в массив V
   i = 0
   For r = 1 To Rcount
      For c = 1 To Ccount
         i = i + 1
         V(r, c) = ValArray(2, i)
      Next c
   Next r
   RandomIntegers = V
End Function
```

Расположение значений диапазона в произвольном порядке

Функция RangeRandomize, представленная ниже, получает в качестве аргумента диапазон и возвращает массив, содержащий этот диапазон с произвольно переставленными значениями.

```
Function RangeRandomize(rng)
   Dim V() As Variant, ValArray() As Variant
   Dim CellCount As Double
   Dim i As Integer, j As Integer
   Dim r As Integer, c As Integer
   Dim Temp1 As Variant, Temp2 As Variant
   Dim RCount As Integer, CCount As Integer

'  Возвращает ошибку, если диапазон слишком большой
   CellCount = rng.Count
   If CellCount > 1000 Then
      RangeRandomize = CVErr(xlErrNA)
      Exit Function
   End If

'  Присваивание значений переменным
   RCount = rng.Rows.Count
   CCount = rng.Columns.Count
   ReDim V(1 To RCount, 1 To CCount)
```

```
  ReDim ValArray(1 To 2, 1 To CellCount)
' Заполнение массива произвольными числами
' и значениями из диапазона
  For i = 1 To CellCount
     ValArray(1, i) = Rnd
     ValArray(2, i) = rng(i)
  Next i
' Сортировка массива ValArray по произвольным числам
  For i = 1 To CellCount
     For j = i + 1 To CellCount
        If ValArray(1, i) > ValArray(1, j) Then
           Temp1 = ValArray(1, j)
           Temp2 = ValArray(2, j)
           ValArray(1, j) = ValArray(1, i)
           ValArray(2, j) = ValArray(2, i)
           ValArray(1, i) = Temp1
           ValArray(2, i) = Temp2
        End If
     Next j
  Next i

' Занесение произвольных значений в массив V
  i = 0
  For r = 1 To Rcount
     For c = 1 To Ccount
        i = i + 1
        V(r, c) = ValArray(2, i)
     Next c
  Next r
  RangeRandomize = V
End Function
```

Этот код подобен коду функции `RandomIntegers`. На рис. 11.18 показана эта функция в действии. В диапазон B2:B11 введена следующая формула массива:

```
{= RangeRandomize(A2:A11)}
```

Эта формула возвращает содержимое диапазона A2:A11, но в случайном порядке.

	A	B
1	**Исходный порядок**	**Случайный порядок**
2	Гиппопотам	Жираф
3	Жираф	Павиан
4	Игуана	Пекари
5	Кенгуру	Кошка
6	Кошка	Кенгуру
7	Лемур	Трубкозуб
8	Лисица	Игуана
9	Павиан	Лемур
10	Пекари	Лисица
11	Слон	Гиппопотам
12	Собака	Собака
13	Трубкозуб	Слон
14		
15		
16		

Рис. 11.18. Функция `RangeRandomize` возвращает содержимое диапазона в случайном порядке

Вызов функций Windows API

Одна из самых важных возможностей VBA — поддержка функций, которые хранятся в динамически подключаемых библиотеках (Dynamic Link Libraries — DLL). В примерах из настоящего раздела демонстрируются самые популярные функции Windows API.

Примечание

Для простоты изложения представленные в этом разделе объявления API-функций могут корректно выполняться только в среде Excel 2010 (32- и 64-разрядная версии). Соответствующие примеры файлов, находящиеся на компакт-диске, совместимы с предыдущими версиями Excel.

Определение связей с файлами

В Windows многие типы файлов ассоциируются с конкретным приложением. Эта связь позволяет загрузить файл в соответствующее приложение (для этого дважды щелкните мышью на файле).

Функция `GetExecutable` вызывает функцию Windows API с целью получить полный путь к приложению, связанному с указанным файлом. Например, в системе находится ряд файлов с расширением `.txt`; вероятно, один такой файл с названием `Readme.txt` в данный момент расположен в папке Windows. Функцию `GetExecutable` можно применять для определения полного пути приложения (которое запускается после двойного щелчка на выбранном файле).

Примечание

Объявления функций Windows API должны находиться в верхней части модуля VBA.

```
Private Declare PtrSafe Function FindExecutableA Lib _
   "shell32.dll" (ByVal lpFile As String, ByVal _
   lpDirectory As String, ByVal lpResult As String) As Long
Function GetExecutable(strFile As String) As String
   Dim strPath As String
   Dim intLen As Integer
   strPath = Space(255)
   intLen = FindExecutableA(strFile, "\", strPath)
   GetExecutable = Trim(strPath)
End Function
```

На рис. 11.19 показан результат вызова функции `GetExecutable`, в качестве аргумента которой используется имя аудиофайла в формате MP3. Функция возвращает полный путь к приложению, которое связано с этим файлом.

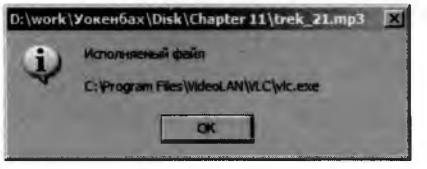


Рис. 11.19. Определение пути и имени для приложения, связанного с заданным файлом

Компакт-диск

Рассматриваемый пример можно найти на прилагаемом к книге компакт-диске в файле `file association.xlsm`.

Определение буквы диска

В VBA нет способа получения информации о дисковых накопителях. Но эта проблема легко решается с помощью трех API-функций, обеспечивающих получение всех необходимых сведений.

На рис. 11.20 представлен результат выполнения процедуры VBA, которая идентифицирует все подключенные дисковые накопители, определяет их тип, а также указывает размер свободного и занятого пространства на диске. Показанная на рисунке система включает шесть дисков.

Столбец1	Столбец2	Столбец3	Столбец4	Столбец5
Диск	Тип	Общий объем	Используемый объем	Свободный объем
A:\	Съемный диск			
C:\	Жесткий диск	80 015 491 072	67 694 669 824	12 320 821 248
D:\	Жесткий диск	160 031 014 912	149 153 247 232	10 877 767 680
E:\	Привод компакт-дисков			
F:\	Привод компакт-дисков			
G:\	Съемный диск	1 029 750 784	19 857 408	1 009 893 376

Рис. 11.20. С помощью функций Windows API можно получить всю информацию о дисках, установленных в системе

Код используемых в примере функций Windows API довольно большой, поэтому он здесь не приводится. Заинтересованные читатели могут найти его на прилагаемом к книге компакт-диске.

Компакт-диск

Рассматриваемый в этом разделе пример находится на прилагаемом к книге компакт-диске в файле `drive information.xlsm`.

Определение параметров принтера по умолчанию

В примере, приведенном в этом разделе, функция Windows API используется для получения информации об активном принтере. Данная информация содержится в одной текстовой строке. Программа анализирует эту строку и отображает информацию в более удобном для чтения формате.

```
Private Declare PtrSafe Function GetProfileStringA Lib _
   "kernel32" (ByVal lpAppName As String, ByVal lpKeyName As _
   String, ByVal lpDefault As String, ByVal lpReturnedString As _
   String, ByVal nSize As Long) As Long
Sub DefaultPrinterInfo()
   Dim strLPT As String * 255
   Dim Result As String
   Call GetProfileStringA _
      ("Windows", "Device", "", strLPT, 254)
   Result = Application.Trim(strLPT)
   ResultLength = Len(Result)
   Comma1 = InStr(1, Result, ",", 1)
   Comma2 = InStr(Comma1 + 1, Result, ",", 1)
'  Получение имени принтера
   Printer = Left(Result, Comma1 - 1)
'  Получение драйвера
   Driver = Mid(Result, Comma1 + 1, Comma2 - Comma1 - 1)
'  Получение последней части строки устройства
   Port = Right(Result, ResultLength - Comma2)
'  Создание сообщения
   Msg = "Принтер:" & Chr(9) & Printer & Chr(13)
   Msg = Msg & "Драйвер:" & Chr(9) & Driver & Chr(13)
   Msg = Msg & "Порт:" & Chr(9) & Port
'  Отображение сообщения
   MsgBox Msg, vbInformation, "Информация о принтере по умолчанию"
End Sub
```

Примечание

Свойство `ActivePrinter` объекта `Application` возвращает название активного принтера (и позволяет его изменить). Но не существует способа определить используемый драйвер принтера или порт. Поэтому эта функция столь полезна.

На рис. 11.21 показано окно сообщения, полученное после выполнения этой процедуры.

Компакт-диск

Этот пример можно найти на прилагаемом к книге компакт-диске в файле `printer info.xlsm`.

Определение текущего видеорежима

В данном примере функции Windows API используются для определения текущего видеорежима системы. Если в приложении необходимо отобразить определенный объем информации на одном экране, то, зная размер экрана, можно правильно задать масштаб текста. Кроме того, в коде определяется количество мониторов в системе. Если установлено более одного монитора, процедура определяет размер виртуального экрана.

```
Declare PtrSafe Function GetSystemMetrics Lib _
   "user32" (ByVal nIndex As Long) As Long
Public Const SM_CMONITORS = 80
Public Const SM_CXSCREEN = 0
Public Const SM_CYSCREEN = 1
Public Const SM_CXVIRTUALSCREEN = 78
Public Const SM_CYVIRTUALSCREEN = 79

Sub DisplayVideoInfo()
   Dim numMonitors As Long
```

```
  Dim vidWidth As Long, vidHeight As Long
  Dim virtWidth As Long, virtHeight As Long
  Dim Msg As String
  numMonitors = GetSystemMetrics(SM_CMONITORS)
  vidWidth = GetSystemMetrics(SM_CXSCREEN)
  vidHeight = GetSystemMetrics(SM_CYSCREEN)
  virtWidth = GetSystemMetrics(SM_CXVIRTUALSCREEN)
  virtHeight = GetSystemMetrics(SM_CYVIRTUALSCREEN)
  If numMonitors > 1 Then
     Msg = numMonitors & " монитора" & vbCrLf
     Msg = Msg & "Виртуальный экран: " & virtWidth & " X "
     Msg = Msg & virtHeight & vbCrLf & vbCrLf
     Msg = Msg & "Видеорежим основного монитора: "
     Msg = Msg & vidWidth & " X " & vidHeight
  Else
     Msg = Msg & "Видеорежим монитора: "
     Msg = Msg & vidWidth & " X " & vidHeight
  End If
  MsgBox Msg
End Sub
```

На рис. 11.22 показано окно сообщения, отображаемое при выполнении процедуры на системе с одним монитором.

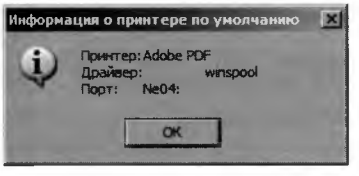


Рис. 11.21. Получение информации об активном принтере с помощью функции Windows API

Рис. 11.22. Использование функций Windows API для определения видеорежима монитора

Компакт-диск

Этот пример доступен на прилагаемом к книге компакт-диске в файле `video mode.xlsm`.

Добавление звука в приложение

Пример, рассматриваемый в этом разделе, расширяет возможности Excel по воспроизведению звука. Таким образом, ваши приложения получают возможность воспроизводить звуковые файлы в форматах WAV и MIDI. Например, при отображении диалогового окна можно воспроизвести небольшой звуковой клип. А можно и не воспроизводить. Как бы там ни было, если возникает необходимость воспроизвести в Excel файлы в формате WAV либо MIDI, изучите этот раздел — в нем содержится нужная информация.

Компакт-диск

Примеры из этого раздела можно найти на прилагаемом к книге компакт-диске в файле `sound.xlsm`.

Воспроизведение WAV-файла

В следующем примере показано объявление API-функции, а также представлена простая процедура — проигрывается файл `sound.wav`, который находится в той же папке, где и рабочая книга.

```
Private Declare Function PlaySound Lib "winmm.dll" _
   Alias "PlaySoundA" (ByVal lpszName As String, _
   ByVal hModule As Long, ByVal dwFlags As Long) As Long
Const SND_SYNC = &H0
Const SND_ASYNC = &H1
Const SND_FILENAME = &H20000

Sub PlayWAV()
   WAVFile = "sound.wav"
   WAVFile = ThisWorkbook.Path & "\" & WAVFile
   Call PlaySound(WAVFile, 0&, SND_ASYNC Or SND_FILENAME)
End Sub
```

В предыдущем примере WAV-файл воспроизводился в асинхронном режиме. Это означает, что выполнение кода продолжается во время воспроизведения звука. Чтобы прекратить выполнение кода во время проигрывания звука, воспользуйтесь следующим оператором:

```
Call PlaySound(WAVFile, 0&, SND_SYNC Or SND_FILENAME)
```

Воспроизведение MIDI-файла

Если звуковой файл имеет формат MIDI, то необходимо применить вызов другой API-функции. MIDI-файлы воспроизводит процедура `PlayMIDI`. Выполнение процедуры `StopMIDI` останавливает проигрывание MIDI-файла. В данном примере применяется файл `xfiles.mid`.

```
Private Declare Function mciExecute Lib "winmm.dll" _
   (ByVal lpstrCommand As String) As Long

Sub PlayMIDI()
   MIDIFile = "xfiles.mid"
   MIDIFile = ThisWorkbook.Path & "\" & MIDIFile
   mciExecute ("play " & MIDIFile)
End Sub

Sub StopMIDI()
   MIDIFile = "xfiles.mid"
   MIDIFile = ThisWorkbook.Path & "\" & MIDIFile
   mciExecute ("stop " & MIDIFile)
End Sub
```

Воспроизведение звука с помощью функции рабочего листа

Функция `Alarm`, показанная ниже, предназначена для применения в формуле рабочего листа. Она использует функцию Windows API для проигрывания звука, если ячейка соответствует определенному условию.

```
Declare Function PlaySound Lib "winmm.dll" _
   Alias "PlaySoundA" (ByVal lpszName As String, _
   ByVal hModule As Long, ByVal dwFlags As Long) As Long
Function Alarm(Cell, Condition)
   Dim WAVFile As String
   Const SND_ASYNC = &H1
   Const SND_FILENAME = &H20000
   If Evaluate(Cell.Value & Condition) Then
      WAVFile = ThisWorkbook.Path & "\sound.wav"
      Call PlaySound(WAVFile, 0&, SND_ASYNC Or SND_FILENAME)
      Alarm = True
   Else
      Alarm = False
```

```
  End If
End Function
```

Функция `Alarm` имеет два аргумента: ссылку на ячейку и "условие" (выраженное в виде строки). Например, следующая формула использует функцию `Alarm` для проигрывания WAV-файла, если значение в ячейке B13 больше или равно 1000.

```
=ALARM(B13,">=1000")
```

Функция использует функцию VBA `Evaluate` для определения, соответствует ли значение ячейки заданному критерию. Если условие выполнено (и звук воспроизведен), функция возвращает ИСТИНА, в противном случае она возвращает значение ЛОЖЬ.

Компакт-диск

Рассмотренные в этом разделе примеры доступны на прилагаемом к книге компакт-диске в файле `sound.xlsm`.

Перекрестная ссылка

Более простой способ использовать звук в функции — воспользоваться описанной ранее функцией `SayIt`.

Чтение и запись параметров системного реестра

Многие приложения Windows используют системный реестр для хранения параметров (о реестре речь шла в главе 4). Процедуры VBA могут считывать значения из реестра и записывать в него новые значения. Для этого необходимы следующие объявления функций Windows API.

```
Private Declare PtrSafe Function RegOpenKeyA Lib "ADVAPI32.DLL" _
  (ByVal hKey As Long, ByVal sSubKey As String, _
  ByRef hkeyResult As Long) As Long
Private Declare PtrSafe Function RegCloseKey Lib "ADVAPI32.DLL" _
  (ByVal hKey As Long) As Long
Private Declare PtrSafe Function RegSetValueExA Lib "ADVAPI32.DLL" _
  (ByVal hKey As Long, ByVal sValueName As String, _

  ByVal dwReserved As Long, ByVal dwType As Long, _
  ByVal sValue As String, ByVal dwSize As Long) As Long
Private Declare PtrSafe Function RegCreateKeyA Lib "ADVAPI32.DLL" _
  (ByVal hKey As Long, ByVal sSubKey As String, _
  ByRef hkeyResult As Long) As Long
Private Declare PtrSafe Function RegQueryValueExA Lib "ADVAPI32.DLL" _
  (ByVal hKey As Long, ByVal sValueName As String, _
  ByVal dwReserved As Long, ByRef lValueType As Long, _
  ByVal sValue As String, ByRef lResultLen As Long) As Long
```

Компакт-диск

Мною разработаны две функции-"оболочки", упрощающие управление реестром: `GetRegistry` и `WriteRegistry`. Их можно найти на прилагаемом к книге компакт-диске в файле `windows registry.xlsm`. В этой рабочей книге также находится процедура, которая демонстрирует запись и чтение данных из системного реестра.

Чтение данных реестра

Функция GetRegistry возвращает раздел из указанного места регистра. Она располагает тремя аргументами.

- RootKey. Строка, представляющая ветвь реестра, к которой обращается функция. Данная строка может принимать одно из следующих значений:
  - HKEY_CLASSES_ROOT;
  - HKEY_CURRENT_USER;
  - HKEY_LOCAL_MACHINE;
  - HKEY_USERS;
  - HKEY_CURRENT_CONFIG;
  - HKEY_DYN_DATA.
- Path. Полный путь к разделу реестра, к которому обращается функция.
- RegEntry. Название параметра, который должна получить функция.

А теперь рассмотрим пример. Если необходимо найти графический файл, используемый в качестве обоев рабочего стола, используйте функцию GetRegistry следующим образом (обратите внимание, что аргументы не чувствительны к регистру).

```
RootKey = "hkey_current_user"
Path = "Control Panel\Desktop"
RegEntry = "Wallpaper"
MsgBox GetRegistry(RootKey, Path, RegEntry), _
   vbInformation, Path & "\RegEntry"
```

После выполнения этой процедуры в окне сообщения отображаются путь и имя графического файла (либо пустая строка, если обои не используются).

Запись данных в реестр

Функция WriteRegistry записывает значение в указанный раздел реестра. Если операция завершается успешно, функция возвращает ИСТИНА; в противном случае функция возвращает ЛОЖЬ. Функция WriteRegistry получает следующие аргументы (все они являются строками).

- RootKey. Строка, представляющая ветвь реестра, к которой обращается функция. Эта строка может принимать одно из перечисленных ниже значений:
  - HKEY_CLASSES_ROOT;
  - HKEY_CURRENT_USER;
  - HKEY_LOCAL_MACHINE;
  - HKEY_USERS;
  - HKEY_CURRENT_CONFIG;
  - HKEY_DYN_DATA.
- Path. Полный путь в реестре. Если пути не существует, он будет создан.
- RegEntry. Название раздела реестра, в который записывается значение. Если раздела не существует, он будет добавлен в реестр.
- RegVal. Значение, которое записывается в реестр.

Ниже приведен пример, в котором записывается значение, представляющее время и дату запуска Excel. Эта информация сохраняется в разделе настроек Excel.

```
Sub Workbook_Open()
   RootKey = "hkey_current_user"
   Path = "software\microsoft\office\14.0\excel\LastStarted"
   RegEntry = "DateTime"
   RegVal = Now()
   If WriteRegistry(RootKey, Path, RegEntry, RegVal) Then
      msg = RegVal & " сохранено в реестре."
         Else msg = "произошла ошибка"
   End If
   MsgBox msg
End Sub
```

Если вы сохраните эту процедуру в личной книге макросов, то указанный параметр будет в дальнейшем автоматически обновляться при каждом запуске Excel.

Простой способ просмотра системного реестра

Если вы решили воспользоваться системным реестром для хранения и выборки настроек приложений Excel, не стоит обращаться к функциям Windows API. Лучше воспользоваться функциями VBA `GetSetting` и `SaveSetting`.

Эти две функции описаны в соответствующем разделе справочной системы, поэтому здесь они подробно не рассматриваются. Важно понимать, что они работают только со следующим разделом реестра:

```
HKEY_CURRENT_USER\Software\VB and VBA Program Settings
```

Другими словами, с помощью этих функций можно управлять данными только одной ветви реестра, в которой сохраняются базовые настройки Excel.

Часть IV

Пользовательские формы

В этой части...

133ак. 3107

Глава 12

Создание собственных диалоговых окон

В этой главе...

- Перед созданием диалоговых окон...
- Использование окон ввода данных
- Функция VBA `MsgBox`
- Метод Excel `GetOpenFilename`
- Метод Excel `GetSaveAsFilename`
- Получение имени папки
- Отображение диалоговых окон Excel
- Отображение формы ввода данных

В этой главе описываются методы создания пользовательских диалоговых окон, которые существенно расширяют стандартные возможности рабочих книг Excel.

Перед созданием диалоговых окон...

Диалоговые окна — это наиболее важный элемент пользовательского интерфейса в Windows. Они применяются практически в каждом приложении Windows, и большинство пользователей неплохо представляет, как они работают. Разработчики Excel создают пользовательские диалоговые окна с помощью пользовательских форм (UserForm). Кроме того, в VBA имеются средства, обеспечивающие создание типовых диалоговых окон.

Перед тем как приступить к изучению тонкостей создания диалоговых окон на основе пользовательских форм, следует научиться использовать некоторые встроенные инструменты Excel, предназначенные для вывода диалоговых окон.

Использование окон ввода данных

Окно ввода данных — это простое диалоговое окно, которое позволяет пользователю ввести одно значение. Например, можно применить окно ввода данных, чтобы предоставить пользователю возможность ввести текст, числа или даже диапазон значений. Для создания окна ввода предназначены две функции `InputBox`: одна — в VBA, а вторая является методом объекта `Application`.

Функция `InputBox` в VBA

Данная функция имеет следующий синтаксис:

```
InputBox(запрос [, заголовок] [, по_умолчанию] [, xpos] [, ypos]
[, справка, раздел])
```

- *Запрос*. Указывает текст, отображаемый в окне ввода (обязательный параметр).
- *Заголовок*. Определяет заголовок окна ввода (необязательный параметр).
- *По_умолчанию*. Задает значение, которое отображается в окне ввода по умолчанию (необязательный параметр).
- *xpos*, *ypos*. Определяют координаты верхнего левого угла окна ввода на экране (необязательные параметры).
- *Справка*, *раздел*. Указывают файл и раздел в справочной системе (необязательные параметры).

Функция `InputBox` запрашивает у пользователя одно значение. Она всегда возвращает строку, поэтому результат нужно будет преобразовать в числовое значение.

Текст, отображаемый в окне ввода, может достигать 1024 символов (длину допускается изменять в зависимости от ширины используемых символов). Кроме того, можно указать заголовок диалогового окна, значение по умолчанию и координаты окна ввода на экране. Также в данном коде указывается раздел справочной системы со всеми вспомогательными сведениями. Если определить этот раздел, то в диалоговом окне будет отображена кнопка Справка.

В следующем примере, показанном на рис. 12.1, используется функция VBA `InputBox`, которая запрашивает у пользователя полное имя (имя и фамилию). Затем программа выделяет имя и отображает приветствие в окне сообщения.

```
Sub GetName()
   Dim UserName As String
   Dim FirstSpace As Integer
   Do Until UserName <> ""
      UserName = InputBox("Укажите имя и фамилию: ", _
         "Представьтесь, пожалуйста")
   Loop
   FirstSpace = InStr(UserName, " ")
   If FirstSpace <> 0 Then
      UserName = Left(UserName, FirstSpace - 1)
   End If
   MsgBox "Привет " & UserName
End Sub
```

Обратите внимание: функция `InputBox` вызывается в цикле `Do Until`. Это позволяет убедиться в том, что данные введены в окно. Если пользователь щелкнет на кнопке Отмена (Cancel) или не введет текст, то переменная `UserName` будет содержать пустую

строку, а окно ввода данных появится повторно. Далее в процедуре будет предпринята попытка получить имя пользователя путем поиска первого символа пробела (для этого применяется функция `InStr`). Таким образом, можно воспользоваться функцией `Left` для получения всех символов, расположенных слева от символа пробела. Если символ пробела не найден, то используется все введенное имя.

Как отмечалось ранее, функция `InputBox` всегда возвращает строку. Если строка, предоставленная в качестве результата выполнения функции `InputBox`, выглядит как число, ее можно преобразовать с помощью функции VBA `Val`. В противном случае следует применить функцию `InputBox` в Excel.

На рис. 12.2 показан еще один пример функции VBA `InputBox`. Пользователю предлагается ввести пропущенное слово. Этот пример также иллюстрирует применение именованных аргументов. Текст запроса выбирается из ячейки рабочего листа.

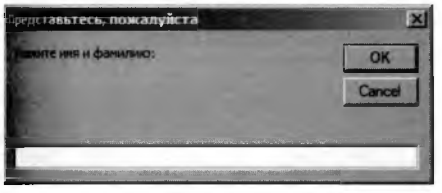


Рис. 12.1. Перед вами — результат выполнения функции VBA `InputBox`

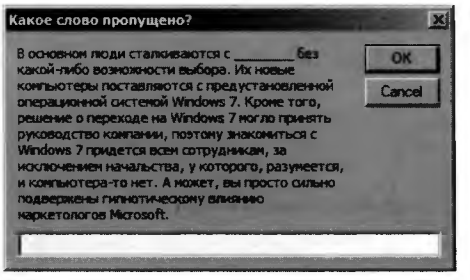


Рис. 12.2. Использование функции VBA `InputBox`, отображающей длинный запрос

```
Sub GetWord()
  Dim TheWord As String
  Dim p As String
  Dim t As String
  p = Range("A1")
  t = "Какое слово пропущено?"
  TheWord = InputBox(prompt:=p, Title:=t)
  If UCase(TheWord) = "WINDOWS 7" Then
     MsgBox "Верно."
  Else
     MsgBox "Неверно."
  End If
End Sub
```

Компакт-диск

Два примера, рассматриваемые в этом разделе, можно найти на прилагаемом к книге компакт-диске в файле VBA `inputbox.xlsm`.

Метод Excel `InputBox`

Метод Excel `InputBox` (по сравнению с функцией VBA `InputBox`) предоставляет три преимущества:

- возможность задать тип возвращаемого значения;
- возможность указать диапазон листа путем выделения с помощью мыши;
- автоматическая проверка правильности введенных данных.

В данном случае метод InputBox имеет следующий синтаксис.

```
InputBox(запрос, [, заголовок], [, по_умолчанию], [, слева],
 [, сверху], [, справка, раздел], [, тип])
```

- *Запрос*. Указывает текст, отображаемый в окне ввода (обязательный параметр).
- *Заголовок*. Определяет заголовок окна ввода (необязательный параметр).
- *По_умолчанию*. Задает значение, которое отображается в окне ввода по умолчанию (необязательный параметр).
- *Слева*, *сверху*. Определяют координаты верхнего левого угла окна ввода на экране (необязательные параметры).
- *Справка*, *раздел*. Указывают файл и раздел в справочной системе (необязательные параметры).
- *Тип*. Указывает код типа данных, который будет возвращаться методом (необязательный параметр). Его возможные значения перечислены в табл. 12.1.

Таблица 12.1. Коды типов данных, возвращаемые методом Excel InputBox

Код	Значение
0	Формула
1	Число
2	Строка (текст)
4	Логическое значение (ИСТИНА или ЛОЖЬ)
8	Ссылка на ячейку как объект диапазона
16	Значение (например, #Н/Д)
64	Массив значений

Метод Excel InputBox достаточно гибок. Используя сумму приведенных выше значений, можно возвратить несколько типов данных. Например, для отображения окна ввода, которое принимает текстовый или числовой тип данных, установите код равным значению 3 (т.е. 1+2 или *число+текст*). Если в качестве кода типа данных применить значение 8, то пользователь сможет ввести в поле адрес ячейки или диапазона ячеек. Кроме того, пользователь имеет возможность указать диапазон на текущем рабочем листе.

В процедуре EraseRange, которая приведена ниже, используется метод InputBox. Таким образом, пользователь может указать удаляемый диапазон (рис. 12.3). Адрес диапазона вводится вручную, мышь необходима для выделения диапазона на листе.

Метод InputBox с кодом 8 возвращает объект Range (обратите внимание на ключевое слово Set). После этого выбранный диапазон очищается (с помощью метода Clear). По умолчанию в поле окна ввода отображается адрес текущей выделенной ячейки. Если в окне ввода щелкнуть на кнопке Отмена (Cancel), то оператор On Error завершит процедуру.

```
Sub EraseRange()
   Dim UserRange As Range
   On Error GoTo Canceled
   Set UserRange = Application.InputBox _
      (Prompt:="Удаляемый диапазон:", _
      Title:="Удаление диапазона", _
      Default:=Selection.Address, _
      Type:=8)
```

```
    UserRange.Clear
    UserRange.Select
    Canceled:
End Sub
```

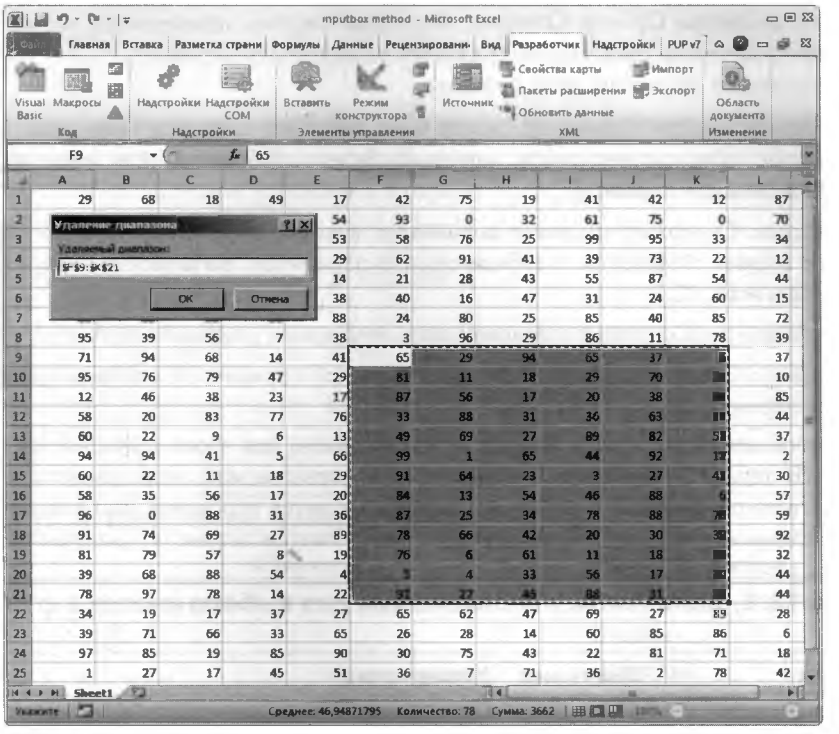


Рис. 12.3. Пример использования метода `InputBox` для определения диапазона

Компакт-диск

Этот пример можно найти на прилагаемом к книге компакт-диске в файле `inputbox method.xlsm`.

Еще одним преимуществом применения метода Excel `InputBox` является автоматическая проверка правильности введенных данных программой Excel. Если в примере `EraseRange` ввести данные, не представляющие диапазон адресов, то Excel отобразит специальное сообщение и позволит пользователю повторить ввод данных (рис. 12.4).

Рис. 12.4. Метод Excel `InputBox` автоматически проверяет вводимые данные

Функция VBA MsgBox

Функция VBA MsgBox предоставляет пользователю простой способ отображения сообщения. Также она задает ответную реакцию пользователя на запрос (передает результат щелчка на кнопке OK или Отмена). Функция MsgBox применяется во многих примерах книги в качестве средства отображения значений переменных.

Ниже приведен синтаксис этой функции.

```
MsgBox(запрос[, кнопки] [, заголовок] [, справка, раздел])
```

- *Запрос*. Определяет текст, который будет отображаться в окне сообщения (обязательный параметр).
- *Кнопки*. Содержит числовое выражение, которое определяет кнопки, отображаемые в окне сообщения (необязательный параметр). Возможные значения приводятся в табл. 12.2.
- *Заголовок*. Содержит заголовок окна сообщения (необязательный параметр).
- *Справка*, *раздел*. Указывают файл и раздел справочной системы (необязательные параметры).

Окна сообщений несложно изменить. Как правило, для этого применяется параметр Кнопки. (В табл. 12.2 приведены константы, которые можно использовать в качестве значений этого параметра.) С его помощью указываются отображаемые кнопки, отмечается необходимость использования значка и определяется кнопка по умолчанию.

Таблица 12.2. Константы, используемые для выбора кнопок в функции MsgBox

Константа	Значение	Назначение
vbOKOnly	0	Отображает только кнопку OK
vbOKCancel	1	Отображает кнопки OK и Отмена
vbAbortRetryIgnore	2	Отображает кнопки Прервать, Повтор и Пропустить
vbYesNoCancel	3	Отображает кнопки Да, Нет и Отмена
vbYesNo	4	Отображает кнопки Да и Нет
vbRetryCancel	5	Отображает кнопки Повтор и Отмена
vbCritical	16	Отображает значок важного сообщения
vbQuestion	32	Отображает значок важного запроса
vbExclamation	48	Отображает значок предупреждающего сообщения
vbInformation	64	Отображает значок информационного сообщения
vbDefaultButton1	0	По умолчанию выделена первая кнопка
vbDefaultButton2	256	По умолчанию выделена вторая кнопка
vbDefaultButton3	512	По умолчанию выделена третья кнопка
vbDefaultButton4	768	По умолчанию выделена четвертая кнопка
vbSystemModal	4096	Выполнение приложений приостанавливается до тех пор, пока пользователь не ответит на запрос в окне сообщения (работает не во всех случаях)
vbMsgBoxHelpButton	16384	Отображает кнопку справки. Имейте в виду, что не существует способа отобразить какой-либо раздел справки после щелчка на этой кнопке

Можно использовать функцию MsgBox в качестве процедуры (для отображения сообщения), а также присвоить возвращаемое этой функцией значение переменной. Функция MsgBox возвращает результат, представляющий кнопку, на которой щелкнул пользователь. В следующем примере отображается сообщение и не возвращается результат.

```
Sub MsgBoxDemo()
   MsgBox "При выполнении макроса ошибок не произошло."
End Sub
```

Чтобы получить результат из окна сообщения, присвойте возвращаемое функцией MsgBox значение переменной. В следующем коде используется ряд встроенных констант (табл. 12.3), которые упрощают управление возвращаемыми функцией MsgBox значениями.

```
Sub GetAnswer()
   Dim Ans As Integer
   Ans = MsgBox("Продолжать?", vbYesNo)
   Select Case Ans
      Case vbYes
'       ...[код при Ans равно Yes]
      Case vbNo
'       ...[код при Ans равно No]
   End Select
End Sub
```

Таблица 12.3. Константы, возвращаемые Msgbox

Константа	Значение	Нажатая кнопка
vbOK	1	OK
vbCancel	2	Отмена
vbAbort	3	Прервать
vbRetry	4	Повтор
vbIgnore	5	Пропустить
vbYes	6	Да
vbNo	7	Нет

Функция MsgBox возвращает переменную, имеющую тип Integer. Вам необязательно использовать переменную для хранения результата выполнения функции MsgBox. Следующая процедура представляет собой вариацию процедуры GetAnswer.

```
Sub GetAnswer2()
   If MsgBox("Продолжать?", vbYesNo) = vbYes Then
'       ...[код при Ans равно Yes]
   Else
'       ...[код при Ans равно No]
   End If
End Sub
```

В следующем примере функции используется комбинация констант для отображения окна сообщения с кнопкой Да, кнопкой Нет и значком вопросительного знака. Вторая кнопка используется по умолчанию (рис. 12.5). Для простоты константы добавлены в переменную Config.

```
Private Function ContinueProcedure() As Boolean
   Dim Config As Integer
   Dim Ans As Integer
   Config = vbYesNo + vbQuestion + vbDefaultButton2
```

```
   Ans = MsgBox("Произошла ошибка. Продолжить?", Config)
   If Ans = vbYes Then ContinueProcedure = True _
      Else ContinueProcedure = False
End Function
```

Функция `ContinueProcedure` может вызываться из другой процедуры. Например, приведенный далее оператор вызывает функцию `ContinueProcedure` (которая отображает окно сообщения). Если функция возвращает значение `ЛОЖЬ` (т.е. пользователь щелкнул на кнопке **Нет**), то процедура будет завершена. В противном случае выполняется следующий оператор:

```
If Not ContinueProcedure() Then Exit Sub
```

Если в сообщении необходимо указать разрыв строки, воспользуйтесь константой `vbCrLf` (или `vbNewLine`) в необходимом месте. Отобразим сообщение в три строки (рис. 12.6).

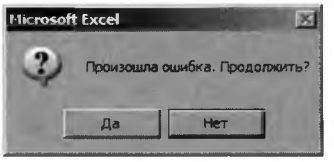


Рис. 12.5. Параметр `Кнопки` функции `MsgBox` определяет кнопки, которые отображаются в окне сообщения

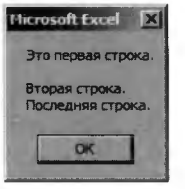


Рис. 12.6. Разбиение сообщения на несколько строк

```
Sub MultiLine()
   Dim Msg As String
   Msg = "Это первая строка." & vbCrLf & vbCrLf
   Msg = Msg & "Вторая строка." & vbCrLf
   Msg = Msg & "Третья строка."
   MsgBox Msg
End Sub
```

Для включения в сообщение символа табуляции применяется константа `vbTab`. В приведенной далее процедуре окно сообщения используется для отображения диапазона значений размером 13×3 — ячейки A1:C13 (рис. 12.7). В этом случае столбцы разделены с помощью константы `vbTab`. Новые строки вставляются с помощью константы `vbCrLf`. Функция `MsgBox` принимает в качестве параметра строку, длина которой не превышает 1023 символов. Такая длина задает ограничение на количество ячеек, которое можно отобразить в сообщении. Обратите внимание, что, поскольку символы табуляции фиксированы, столбцы не выравниваются в случае, если ячейка содержит более 11 символов.

```
Sub ShowRange()
   Dim Msg As String
   Dim r As Integer, c As Integer
   Msg = ""
   For r = 1 To 12
      For c = 1 To 3
         Msg = Msg & Cells(r, c).Text
         If c <> 3 Then Msg = Msg & vbTab
      Next c
      Msg = Msg & vbCrLf
   Next r
   MsgBox Msg
End Sub
```

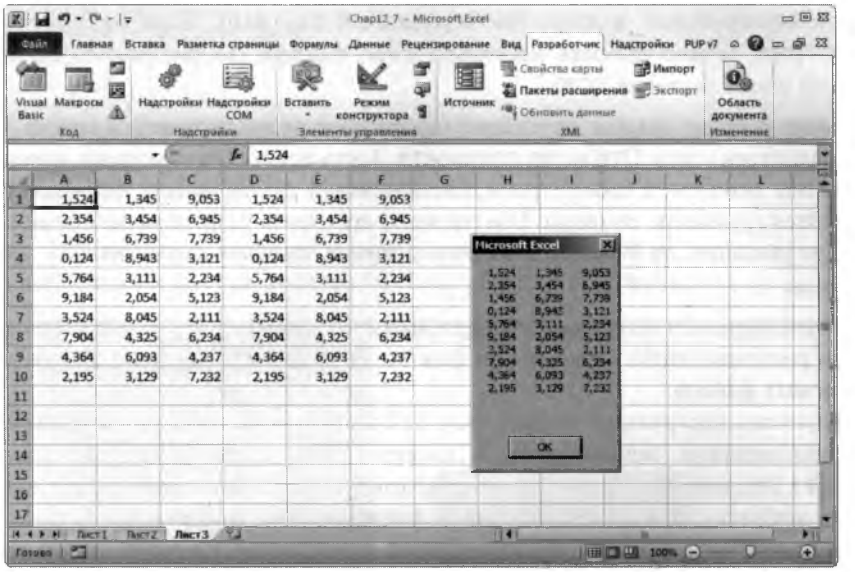


Рис. 12.7. Текст в этом окне сообщения содержит символы табуляции и разрыва строк

Перекрестная ссылка

В главе 15 приводится пример пользовательской формы, которая имитирует работу функции `MsgBox`.

Метод Excel `GetOpenFilename`

Если приложению необходимо получить от пользователя имя файла, то можно воспользоваться функцией `InputBox`, но этот подход часто приводит к возникновению ошибок. Более надежным считается использование метода `GetOpenFilename` объекта `Application`, который позволяет удостовериться, что приложение получило корректное имя файла (а также его полный путь).

Данный метод позволяет отобразить стандартное диалоговое окно **Открытие документа**, но при этом указанный файл не открывается. Вместо этого метод возвращает строку, которая содержит путь и имя файла, выбранные пользователем. По окончании данного процесса с именем файла можно делать все что угодно.

Этот метод имеет следующий синтаксис (все параметры необязательные):

```
Application.GetOpenFilename(фильтр_файла, индекс_фильтра, заголовок,
подпись_кнопки, множественный_выбор)
```

- *`Фильтр_файла`*. Содержит строку, определяющую критерий фильтрации файлов (необязательный параметр).
- *`Индекс_фильтра`*. Указывает индексный номер того критерия фильтрации файлов, который используется по умолчанию (необязательный параметр).
- *`Заголовок`*. Содержит заголовок диалогового окна (необязательный параметр). Если этот параметр не указать, то будет использован заголовок **Открытие документа**.
- *`Подпись_кнопки`*. Применяется только в компьютерах Macintosh.

- *Множественный_выбор*. Необязательный параметр. Если он имеет значение ИСТИНА, можно выбрать несколько имен файлов. По умолчанию данный параметр имеет значение ЛОЖЬ.

Аргумент *фильтр_файла* определяет содержимое раскрывающегося списка Тип файлов, находящегося в окне Открытие документа. Аргумент состоит из строки, определяющей отображаемое значение, а также строки действительной спецификации типа файлов, в которой находятся групповые символы. Оба элемента аргумента разделены запятыми. Если этот аргумент не указывать, то будет использовано значение, заданное по умолчанию.

```
"Все файлы (*.*),*.*"
```

Обратите внимание на первую часть строки `Все файлы (*.*)`. Это текст, отображаемый в раскрывающемся списке тип файлов. Вторая часть строки `*.*` указывает тип отображаемых файлов.

В следующих инструкциях переменной `Filt` присваивается строковое значение. Эта строка впоследствии используется в качестве аргумента *фильтр_файла* метода `GetOpenFilename`. В данном случае диалоговое окно предоставит пользователю возможность выбрать один из четырех типов файлов (кроме варианта `Все файлы`). Если задать значение переменной `Filt`, то будет использоваться оператор конкатенации строки VBA. Этот способ упрощает управление громоздкими и сложными аргументами.

```
Filt = "Текстовые файлы (*.txt),*.txt," & _
   "Файлы Lotus (*.prn),*.prn," & _
   "Файлы, разделенные запятой (*.csv),*.csv," & _
   "Файлы ASCII (*.asc),*.asc," & _
"Все файлы (*.*),*.*"
```

Аргумент *индекс_фильтра* указывает значение аргумента *фильтр_файла*, заданное по умолчанию. Аргумент *заголовок* определяет текст, который отображается в заголовке окна. Если параметр *множественный_выбор* имеет значение ИСТИНА, то пользователь может выбрать в окне несколько файлов. Имя каждого файла заносится в массив.

В следующем примере у пользователя запрашивается имя файла. При этом в поле типа файлов используются пять фильтров.

```
Sub GetImportFileName()
   Dim Filt As String
   Dim FilterIndex As Integer
   Dim Title As String
   Dim FileName As Variant

'  Настройка списка фильтров
   Filt = "Текстовые файлы (*.txt),*.txt," & _
      "Файлы Lotus (*.prn),*.prn," & _
      "Файлы, разделенные запятой (*.csv),*.csv," & _
      "Файлы ASCII (*.asc),*.asc," & _
      "Все файлы (*.*),*.*"
'  По умолчанию используется фильтр *.*
   FilterIndex = 5
'  Заголовок окна
   Title = "Выберите импортируемый файл"
'  Получение имени файла
   FileName = Application.GetOpenFilename _
      (FileFilter:=Filt, _
      FilterIndex:=FilterIndex, _
      Title:=Title)
```

```
' При отмене выйти из окна
  If FileName = False Then
     MsgBox "Файл не выбран."
     Exit Sub
  End If

' Отображение полного пути и имени файла
  MsgBox "Выбран файл " & FileName
End Sub
```

На рис. 12.8 показано диалоговое окно, которое выводится на экран после выполнения этой процедуры и выбора фильтра "Файлы, разделенные запятой".

Рис. 12.8. Метод `GetOpenFilename` отображает диалоговое окно, в котором выбирается файл

Приведенный далее пример напоминает предыдущий. Разница заключается в том, что пользователь может, удерживая нажатыми клавиши <Shift> и <Ctrl>, выбрать в окне несколько файлов. Обратите внимание, что событие использования кнопки Отмена определяется по наличию переменной массива `FileName`. Если пользователь не щелкнул на кнопке Отмена, то результирующий массив будет состоять как минимум из одного элемента. В этом примере список выбранных файлов отображается в окне сообщения.

```
Sub GetImportFileName2()
   Dim Filt As String
   Dim FilterIndex As Integer
   Dim FileName As Variant
   Dim Title As String
   Dim i As Integer
   Dim Msg As String

' Настройка фильтров файлов
   Filt = "Текстовые файлы (*.txt),*.txt," & _
          "Файлы Lotus (*.prn),*.prn," & _
          "Файлы, разделенные запятыми (*.csv),*.csv," & _
          "Файлы ASCII (*.asc),*.asc," & _
          "Все файлы (*.*),*.*"

' По умолчанию используется фильтр *.*
   FilterIndex = 5
' Заголовок диалогового окна
   Title = "Выбор импортируемого файла"
```

```
'  Получение имени файла
   FileName = Application.GetOpenFilename _
      (FileFilter:=Filt, _
      FilterIndex:=FilterIndex, _
      Title:=Title, _
      MultiSelect:=True)

'  Закрытие окна при использовании кнопки "Отмена"
   If Not IsArray(FileName) Then
      MsgBox "Файлы не выбраны."
      Exit Sub
   End If


'  Отображение полного пути и имени файла
   For i = LBound(FileName) To UBound(FileName)
      Msg = Msg & FileName(i) & vbCrLf
   Next i
   MsgBox "Вы выбрали:" & vbCrLf & Msg
End Sub
```

Обратите внимание: переменная `FileName` определена как массив переменного типа (а не как строка в предыдущем примере). Причина заключается в том, что потенциально `FileName` может содержать массив значений, а не только одну строку.

Компакт-диск

Два рассмотренных в этом разделе примера доступны на прилагаемом к книге компакт-диске в файле `prompt for file.xlsm`.

Метод Excel `GetSaveAsFilename`

Данный метод имеет много общего с методом `GetOpenFilename`. Он отображает диалоговое окно Сохранение документа и дает пользователю возможность выбрать (или указать) имя сохраняемого файла. В результате возвращается имя файла, но никакие действия не предпринимаются.

Этот метод имеет следующий синтаксис:

```
Application.GetSaveAsFilename(начальное_имя, фильтр_файла,
индекс_фильтра, заголовок, текст_кнопки)
```

Ниже описаны аргументы метода.

- *Начальное_имя*. Указывает предполагаемое имя файла (необязательный параметр).
- *Фильтр_файла*. Содержит критерий фильтрации отображаемых в окне файлов (необязательный параметр).
- *Индекс_фильтра*. Код критерия фильтрации файлов, который используется по умолчанию (необязательный параметр).
- *Заголовок*. Определяет текст заголовка диалогового окна (необязательный параметр).
- *Текст_кнопки*. Предназначен только для платформ Macintosh.

Получение имени папки

Для того чтобы получить имя файла, проще всего воспользоваться описанным выше методом `GetOpenFileName`. Но если нужно получить лишь имя папки (без названия файла), лучше воспользоваться методом объекта Excel `FileDialog`.

Следующая процедура отображает диалоговое окно (рис. 12.9), в котором можно выбрать папку. Затем с помощью функции `MsgBox` отображается имя выбранной папки (или сообщение `Отменено`).

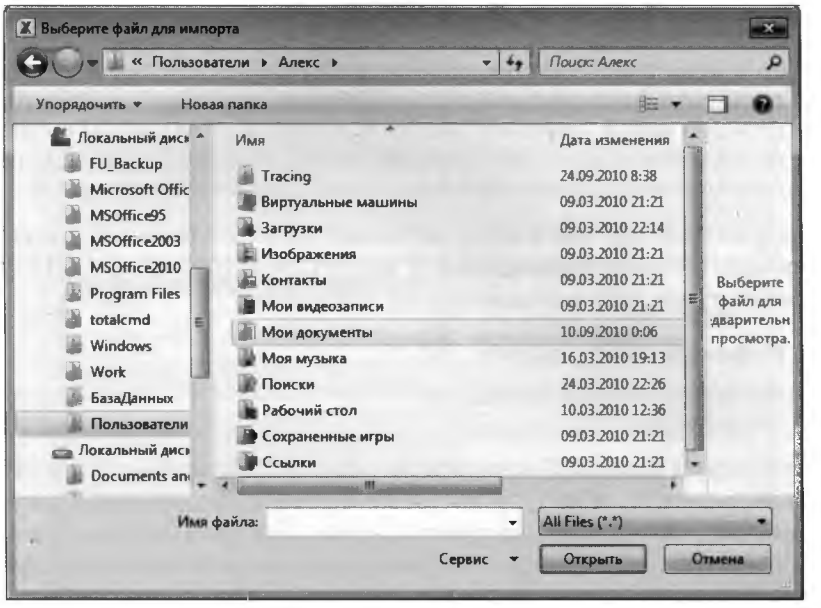


Рис. 12.9. Пример использования объекта `FileDialog` для выбора папки

```
Sub GetAFolder ()
  With Application.FileDialog(msoFileDialogFolderPicker)
     .InitialFileName = Application.DefaultFilePath & "\"
     .Title = "Выберите место хранения резервной копии"
     .Show
     If .SelectedItems.Count = 0 Then
        MsgBox "Отменено"
     Else
        MsgBox .SelectedItems(1)
     End If
  End With
End Sub
```

Объект `FileDialog` позволяет определить начальную папку путем указания значения свойства `InitialFileName`. В рассматриваемом примере в качестве начальной папки применяется путь к файлам Excel, заданный по умолчанию.

Отображение диалоговых окон Excel

Создаваемый вами код VBA может вызывать на выполнение многие команды Excel, находящиеся на ленте. И если в результате выполнения команды открывается диалоговое

окно, ваш код может делать выбор в диалоговом окне (даже если само диалоговое окно не отображается). Например, следующая инструкция VBA эквивалентна выбору команды Главная⇨Редактирование⇨Найти и выделить⇨Перейти (Home⇨Editing⇨Find & Select⇨Go To) и указанию диапазона ячеек A1:C3 с последующим щелчком на кнопке OK. Но само диалоговое окно Переход (Go To) при этом не отображается (именно это и нужно).

```
Application.Goto Reference:=Range("A1:C3")
```

Иногда же приходится отображать встроенные окна Excel, чтобы пользователь мог сделать свой выбор. Для этого используется коллекция `Dialogs` объекта `Application`.

Примечание

Еще один способ отображения встроенных диалоговых окон Excel — воспользоваться коллекцией `Dialogs` объекта `Application`. Учтите, что в настоящее время компания Microsoft прекратила поддержу этого свойства, поэтому оно даже не обсуждается. Альтернативное решение, обеспечивающее отображение диалоговых окон Excel, рассматривается ниже.

В предыдущих версиях Excel пользовательские меню и панели инструментов создавались с помощью объекта `CommandBar`. В версиях Excel 2007 и Excel 2010 этот объект по-прежнему доступен, хотя и работает не так, как раньше.

Перекрестная ссылка

Дополнительные сведения, относящиеся к объекту `CommandBar`, можно найти в главе 22.

Начиная с версии Excel 2007 возможности объекта `CommandBar` были существенно расширены. В частности, объект `CommandBar` можно использовать для вызова команд ленты с помощью VBA. Многие из команд, доступ к которым открывается с помощью ленты, отображают диалоговое окно. Например, следующая инструкция отображает диалоговое окно Вывод на экран скрытого листа (рис. 12.10).

```
Application.CommandBars.ExecuteMso("SheetUnhide")
```

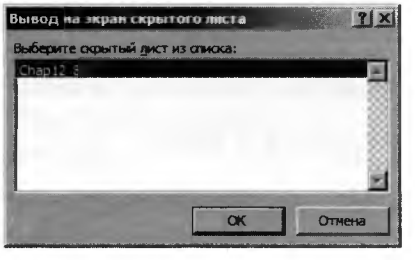


Рис. 12.10. Диалоговое окно, отображаемое в результате выполнения указанного выше оператора VBA

Метод `ExecuteMso` принимает лишь один аргумент, `idMso`, который представляет элемент управления ленты. К сожалению, сведения о многих параметрах в справочной системе отсутствуют.

Компакт-диск

На прилагаемом компакт-диске находится рабочая книга `ribbon control names.xlsx`, в которой описаны все названия параметров команд ленты Excel. Поэкспериментируйте с параметрами, перечисленными в этой рабочей книге. Многие из них вызывают команды немедленно (без промежуточных диалоговых окон). Но большинство из них генерирует ошибку при использовании в неправильном контексте. Например, Excel отображает сообщение об ошибке, если команда `FunctionWizard` вызывается в случае выбора диаграммы.

А теперь рассмотрим еще один пример. В результате выполнения приведенного ниже оператора отображается вкладка Шрифт диалогового окна Формат ячеек (рис. 12.11).

```
Application.CommandBars.ExecuteMso("FormatCellsFontDialog")
```

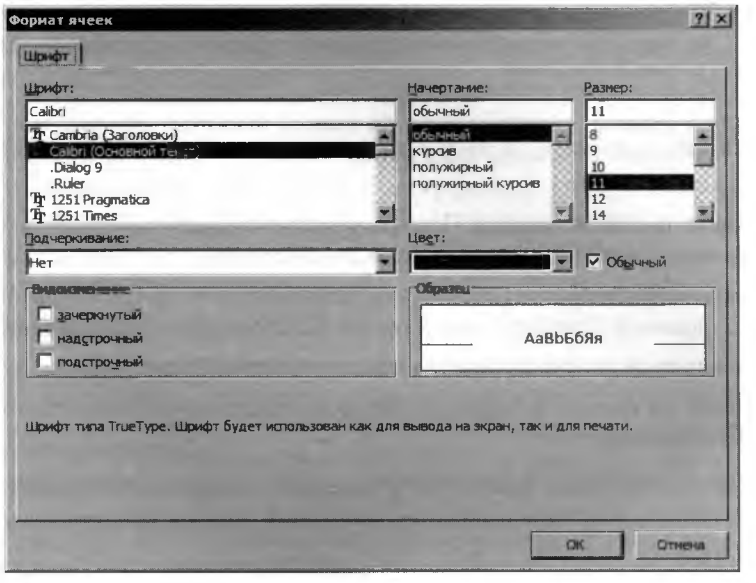


Рис. 12.11. Использование метода `ExecuteMso` для отображения диалогового окна

Отображение формы ввода данных

Многие пользователи применяют Excel для управления списками, информация в которых ранжирована по строкам. В Excel поддерживается простой способ работы с подобными типами данных с помощью встроенных форм ввода данных, которые могут создаваться автоматически. Подобная форма предназначена для работы как с обычным диапазоном, так и с диапазоном, оформленным в виде таблицы (с помощью команды Вставка⇨Таблицы⇨Таблица (Insert⇨Tables⇨Table)). Пример формы ввода данных показан на рис. 12.12.

Доступ к формам ввода данных

В силу каких-то неизвестных причин на ленте Excel отсутствует команда, обеспечивающая доступ к форме ввода данных. Подобную команду можно добавить на панель быстрого доступа. Для этого выполните следующие действия.

Рис. 12.12. Некоторые пользователи предпочитают применять встроенные формы ввода данных Excel для ввода сведений

1. Щелкните правой кнопкой мыши на панели быстрого доступа и в контекстном меню выберите параметр **Настройка панели быстрого доступа** (Customize Quick Access Toolbar).

   На экране появится вкладка **Панель быстрого доступа** (Quick Access Toolbar) диалогового окна **Параметры Excel** (Excel Options).

2. В раскрывающемся списке **Выбрать команды из** (Choose Commands From) выберите параметр **Команды не на ленте** (Commands Not in the Ribbon).

3. В появившемся списке выберите параметр **Форма** (Form).

4. Щелкните на кнопке **Добавить** (Add) для добавления выбранной команды на панель быстрого доступа.

5. Щелкните на кнопке **OK** для закрытия диалогового окна **Параметры Excel**.

   После выполнения перечисленных выше действий на панели быстрого доступа появится новый значок.

Доступ к меню, оформленным в стиле Excel 2003

Еще один способ отображения встроенных диалоговых окон требует знания методики работы с панелями инструментов, применяемыми в версиях Excel, предшествующих Excel 2007 (эти панели инструментов также называются объектами `CommandBar`). Хотя в Excel объекты `CommandBar` больше не используются, они по-прежнему поддерживаются для обеспечения совместимости.

Например, следующий оператор эквивалентен выбору пункта Формат⇨Лист⇨Отобразить (Format⇨Sheet⇨Unhide) в меню Excel 2003.

```
Application.CommandBars("Worksheet Menu Bar") _
Controls("Формат").Controls("Лист") _
Controls("Отобразить...").Execute
```

В результате вызова этого оператора отображается диалоговое окно Отобразить (Unhide). Обратите внимание на необходимость точного соответствия заголовков элементов меню (включая многоточие после Отобразить).

Ниже приводится еще один пример. Этот оператор отображает диалоговое окно Формат ячеек (Format Cells).

```
Application.CommandBars("Worksheet Menu Bar") _
Controls("Формат").Controls("Ячейки...").Execute
```

На самом деле пользоваться объектами `CommandBar` не стоит, поскольку вряд ли они будут поддерживаться в будущих версиях Excel.

Для работы с формой ввода данных следует структурировать данные таким образом, чтобы Excel распознавала их в виде таблицы. Начните с указания заголовков столбцов в первой строке диапазона вводимых данных. Выделите любую ячейку в таблице и щелкните на кнопке **Форма** (Form) панели быстрого доступа. Затем Excel отображает диалоговое окно, в котором будут вводиться данные. Для перемещения между текстовыми полями в целях ввода информации используйте клавишу <Tab>. Если ячейка содержит формулу, результат вычислений отображается в виде текста (а не в формате поля ввода данных). Другими словами, невозможно изменить формулы с помощью формы ввода данных.

По завершении ввода данных в форму щелкните на кнопке **Создать** (New). После этого Excel вводит данные в строку рабочего листа, а также очищает диалоговое окно для ввода следующей строки данных.

Отображение формы ввода данных с помощью VBA

Используйте метод `ShowDataForm` для отображения формы ввода данных Excel. Единственное требование заключается в том, что активная ячейка должна находиться в диапазоне. Следующий код активизирует ячейку A1 (в таблице), а затем отображает форму ввода данных.

```
Sub DisplayDataForm()
   Range("A1").Select
   ActiveSheet.ShowDataForm
End Sub
```

Компакт-диск

Рассмотренный пример находится в рабочей книге на прилагаемом компакт-диске (файл `data form example.xlsm`).

Глава 13

Работа с пользовательскими формами

В этой главе...

- Обработка пользовательских диалоговых окон в Excel
- Вставка новой формы `UserForm`
- Добавление элементов управления в пользовательское диалоговое окно
- Элементы управления в окне Toolbox
- Настройка элементов управления пользовательского диалогового окна
- Изменение свойств элементов управления
- Отображение пользовательского диалогового окна
- Закрытие пользовательского диалогового окна
- Пример создания пользовательского диалогового окна
- События объекта `UserForm`
- Ссылка на элементы управления пользовательского диалогового окна
- Настройка панели инструментов Toolbox
- Создание шаблонов диалоговых окон
- Вопросы для самоконтроля

В этой главе рассматриваются методы создания пользовательских форм и работы с ними.

Обработка пользовательских диалоговых окон в Excel

Разрабатывая приложения в Excel, можно относительно просто создавать собственные диалоговые окна. При этом несложно повторить внешний вид и поведение практически всех стандартных диалоговых окон Excel.

Разработчики приложений Excel всегда имели возможность создавать собственные диалоговые окна. Причем, начиная с Excel 97, применяемые при этом процедуры заметно изменились. Пользовательские диалоговые окна заменили неуклюжие диалоговые листы, и у разработчиков появилось намного больше возможностей по управлению собственными диалоговыми окнами. Но в целях совместимости Excel 97 и более поздние версии все еще поддерживают старые диалоговые листы Excel 5/95. Хорошей новостью является то, что формами `UserForm` управлять намного проще; кроме того, они предоставляют широкий набор новых возможностей.

Пользовательские диалоговые окна создаются на основе технологии пользовательских форм, к которым можно получить доступ из редактора Visual Basic (VBE).

Ниже приведена стандартная последовательность действий, которой следует придерживаться при создании пользовательского диалогового окна.

1. Вставьте новую форму `UserForm` в проект VBAProject рабочей книги.
2. Добавьте элементы управления в форму `UserForm`.
3. Настройте свойства добавленных элементов управления.
4. Создайте процедуры "обработчики событий" для элементов управления.

   Эти процедуры добавляются в модуль кода `UserForm` и выполняются при возникновении различных событий (например, при щелчке на кнопке).
5. Разработайте процедуру, которая отображает форму `UserForm`.

   Эта процедура находится в модуле VBA (а не в модуле кода для формы `UserForm`).
6. Определите простой способ вызова на выполнение процедуры, созданной в п. 5.

   Можно поместить кнопку на рабочий лист, команду ленты и т.д.

Вставка новой формы `UserForm`

Для того чтобы добавить в проект форму `UserForm`, запустите VBE (нажмите клавиши <Alt+F11>, укажите рабочую книгу в окне проекта и выполните команду Insert⇨UserForm (Вставка⇨Форма UserForm)). Формы `UserForm` получат такие имена, как `UserForm1`, `UserForm2` и т.д.

Совет

В целях упрощения идентификации можно изменить название формы `UserForm`. Для этого выберите форму и измените свойство Name (Имя) в окне Properties (Свойства). (Если это окно не отображается, нажмите клавишу <F4>.) На рис. 13.1 показано окно Properties, в котором выбрана пустая форма `UserForm`.

В рабочей книге может быть произвольное количество форм `UserForm`, а каждая форма включает единственное пользовательское диалоговое окно.

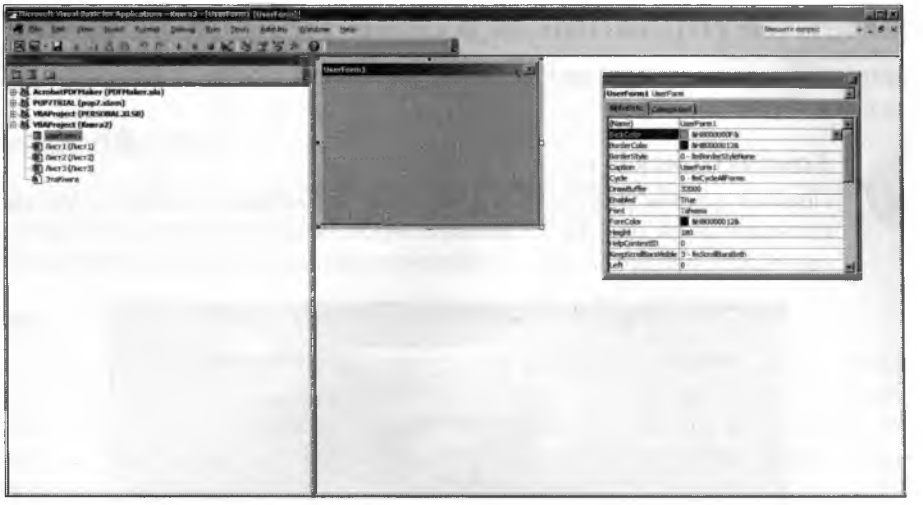


Рис. 13.1. Окно Properties для пустой формы `UserForm`

Добавление элементов управления в пользовательское диалоговое окно

Чтобы добавить элементы управления в форму `UserForm`, воспользуйтесь панелью Toolbox (Набор инструментов). Обратите внимание, что в VBE отсутствуют команды меню, предназначенные для добавления элементов управления. Если панель Toolbox не отображена на экране, выберите команду View⇨Toolbox (Вид⇨Панель инструментов Toolbox). Панель Toolbox показана на рис. 13.2.

Рис. 13.2. Воспользуйтесь окном Toolbox для добавления элементов управления в пользовательскую форму

Щелкните на той кнопке в панели Toolbox, которая соответствует добавляемому элементу управления. После этого щелкните внутри диалогового окна для создания элемента управления (используется размер элемента по умолчанию). Также можно щелкнуть на элементе управления и, перетащив его границы в диалоговом окне, задать необходимый размер в пользовательском диалоговом окне.

Добавленному элементу управления назначается имя, которое состоит из названия типа элемента управления и числового кода. Например, если добавить элемент управления `CommandButton` в пустую форму `UserForm`, то этот элемент управления будет называться `CommandButton1`. Если добавить в окно второй элемент управления `CommandButton`, то он будет называться `CommandButton2`.

Совет

Рекомендуется переименовать все элементы управления, управление которыми происходит с помощью кода VBA. Это позволит воспользоваться описательными именами, простыми для запоминания. Согласитесь, что `ProductListBox` звучит гораздо лучше, чем `ListBox1`. Для изменения названия элемента управления воспользуйтесь окном Properties (Свойства) в VBE. Просто выберите объект и измените его свойство Name.

Элементы управления в окне Toolbox

В следующих разделах описаны элементы управления, доступ к которым можно получить в окне Toolbox.

Компакт-диск

Пример формы `UserForm`, на которой размещены элементы управления, приводится на рис. 13.3. Соответствующая рабочая книга называется `all userform controls.xlsm` и находится на прилагаемом компакт-диске.

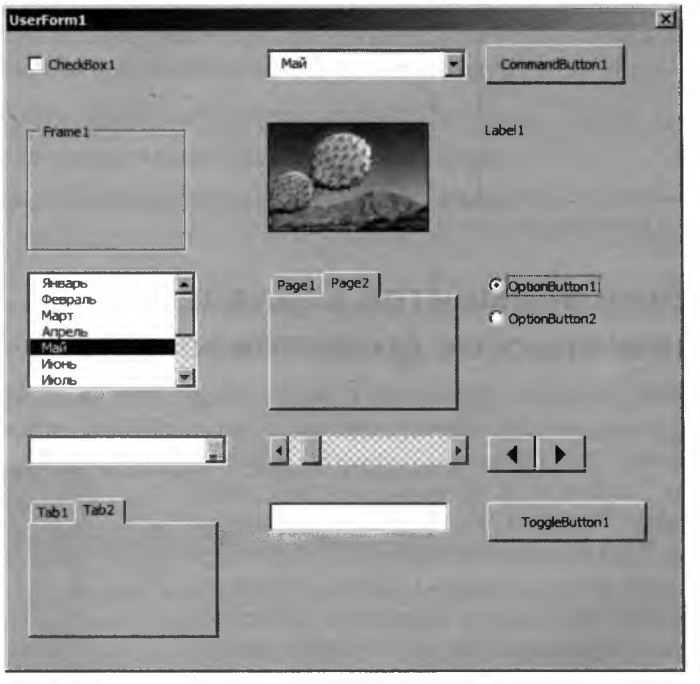


Рис. 13.3. Эта форма `UserForm` содержит 15 элементов управления

Совет

Формы `UserForm` могут также использовать другие элементы управления ActiveX (см. раздел "Настройка панели инструментов Toolbox").

CheckBox

Элемент управления `CheckBox` предоставляет пользователю возможность выбрать один из двух вариантов: да или нет, истина или ложь, включить или выключить и т.д. Если элемент управления `CheckBox` установлен, то он имеет значение `True`, в противном случае значение равно `False`.

ComboBox

Элемент управления `ComboBox` подобен объекту `ListBox`. Отличие заключается в том, что `ComboBox` представляет раскрывающийся список, в котором в определенный

момент времени отображается только одно значение. Кроме того, пользователю в поле списка разрешено вводить значение, которое необязательно представляет одну из опций объекта ComboBox.

CommandButton

Каждое создаваемое диалоговое окно будет иметь как минимум один элемент управления CommandButton. Обычно используются объекты CommandButton, представляющие кнопки OK и Отмена.

Frame

Элемент управления Frame применяется в качестве оболочки для других элементов управления. Он добавляется в диалоговое окно либо в целях эстетики, либо из соображений логического группирования однотипных элементов управления. Элемент управления Frame требуется в случае, если в диалоговом окне содержится более одного набора элементов управления OptionButton.

Image

Элемент управления Image используется для представления графического изображения, которое сохранено в отдельном файле или вставляется из буфера обмена. Кроме того, элемент управления Image незаменим при отображении в диалоговом окне логотипа компании. Графическое изображение сохраняется вместе с рабочей книгой. Таким образом, вместе с рабочей книгой передавать другому пользователю копию графического файла необязательно.

Предупреждение

Некоторые графические файлы занимают много места, поэтому их включение в рабочую книгу приведет к радикальному увеличению ее размера. Поэтому используйте небольшие по размеру картинки либо вовсе откажитесь от них.

Label

Элемент управления Label отображает текст в диалоговом окне.

ListBox

Элемент управления ListBox предоставляет список опций, из которого пользователь может выбрать один вариант (или несколько). Элемент управления ListBox невероятно гибок в использовании. Например, вы вправе указать диапазон на листе, который содержит элементы списка. Этот диапазон может состоять из нескольких столбцов. Кроме того, элемент управления ListBox заполняется опциями с помощью кода VBA.

MultiPage

Элемент управления MultiPage позволяет создавать диалоговые окна с несколькими вкладками, которые подобны появляющимся после выбора команды Формат ячеек (Format Cells). По умолчанию элемент управления MultiPage состоит из двух вкладок.

OptionButton

Элемент управления OptionButton применяется при выборе пользователем одного варианта из нескольких. Эти элементы управления всегда группируются в диалоговом окне в наборы, содержащие не менее двух опций. Когда один элемент управления OptionButton выбран, все остальные элементы управления OptionButton текущей группы автоматически становятся неактивными.

Если в пользовательском диалоговом окне содержится более одного набора элементов управления OptionButton, то каждый из таких наборов должен иметь собственное значение свойства GroupName. В противном случае все элементы управления OptionButton в диалоговом окне рассматриваются как члены одной группы. Также можно вставить элементы управления OptionButton в объект Frame, что приведет к их автоматическому группированию в текущем разделе.

RefEdit

Элемент управления RefEdit используется тогда, когда пользователь должен выделить диапазон ячеек на листе.

ScrollBar

Элемент управления ScrollBar в некотором смысле подобен элементу управления SpinButton. Разница заключается в том, что пользователь может перетаскивать ползунок объекта ScrollBar для изменения значения с большим приращением. Элемент управления ScrollBar рекомендуется использовать при выборе значения из большого диапазона.

SpinButton

Элемент управления SpinButton позволяет выбрать значение после щелчка на одной из двух кнопок со стрелками. Одна из них применяется для увеличения значения, а вторая — для уменьшения. Элемент управления SpinButton часто используется совместно с элементами управления TextBox и Label, которые содержат текущее значение элемента управления SpinButton.

TabStrip

Элемент управления TabStrip подобен элементу управления MultiPage, однако использовать его сложнее. Элемент управления TabStrip, в отличие от MultiPage, не выступает контейнером для других объектов. Как правило, элемент управления MultiPage обладает более широкими возможностями.

TextBox

Элемент управления TextBox позволяет пользователям вводить текст в диалоговом окне.

ToggleButton

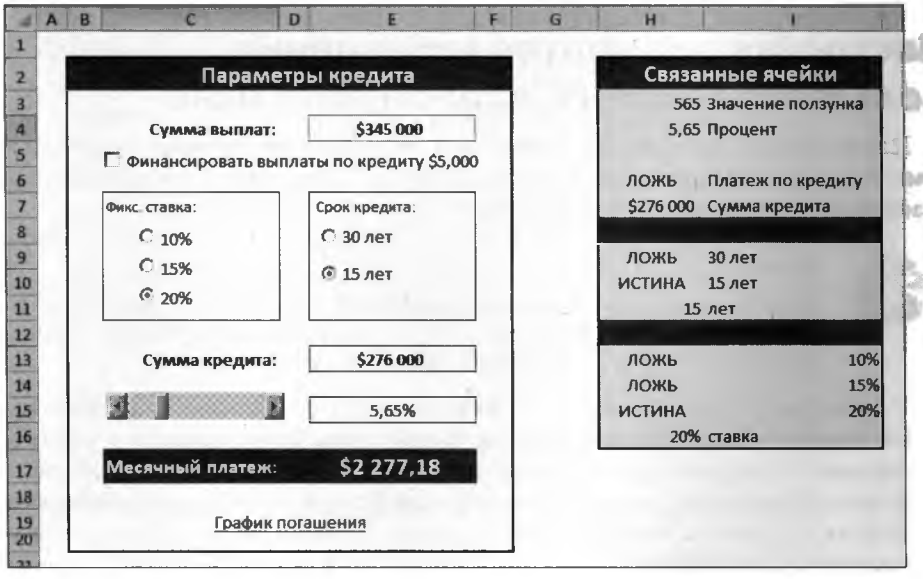
Элемент управления ToggleButton имеет два состояния: включен и выключен. Щелчок на кнопке приводит к изменению состояния на противоположное и к изменению внешнего вида кнопки. Этот элемент управления может иметь значение True (активен) или False (неактивен). Он не относится к "стандартным" элементам управления, по-

этому использование двух элементов управления OptionButton или одного CheckBox может оказаться более удачным вариантом.

Использование элементов управления на рабочем листе

Многие из элементов управления пользовательскими диалоговыми окнами могут встраиваться непосредственно в рабочий лист. Доступ к этим элементам можно получить также путем выполнения команды Excel Разработчик⇨Элементы управления⇨Вставить (Developer⇨Controls⇨Insert). Для использования подобных элементов в составе рабочего листа требуется гораздо меньше усилий, чем для создания пользовательского диалогового окна. Кроме того, в данном случае можно не создавать макросы, поскольку элемент управления можно связать с ячейкой рабочего листа. Например, если на рабочий лист вставить элемент управления CheckBox (Флажок), его можно связать с нужной ячейкой, задав свойство LinkedCell. Если флажок установлен, в связанной ячейке отображается значение ИСТИНА. Если же флажок сброшен, то в связанной ячейке отображается значение ЛОЖЬ.

На представленном ниже рисунке отображается рабочий лист, содержащий некоторые элементы управления ActiveX. Эта рабочая книга, называемая activex worksheet controls.xlsx, находится на прилагаемом компакт-диске. Она включает связанные ячейки и не содержит макросов.



Добавление элементов управления в рабочий лист может оказаться непростой задачей, поскольку они могут происходить из двух источников.

- **Элементы управления формами.** Эти элементы управления являются внедряемыми объектами.
- **Элементы управления ActiveX.** Эти элементы управления являются подмножеством элементов, доступных в пользовательских диалоговых окнах.

Можно использовать элементы управления, относящиеся к одному из двух вышеперечисленных источников, но при этом следует понимать различия между ними. Эле-

менты управления пользовательскими диалоговыми окнами работают не так, как элементы управления ActiveX.

После добавления элемента управления ActiveX в рабочий лист Excel переходит в режим конструктора. В этом режиме можно настраивать свойства любого элемента управления рабочего листа, добавлять или изменять процедуры обработки событий для элемента управления, а также изменять его размер или положение. Для отображения окна свойств (Properties) элемента управления ActiveX воспользуйтесь командой Разработчик⇨Элементы управления⇨Свойства (Developer⇨Controls⇨Properties).

Для создания простых кнопок можно использовать элемент управления `Button` (Кнопка), который находится на панели инструментов Формы (Form). В этом случае обеспечивается возможность запуска макроса. Если же воспользоваться элементом управления `CommandButton`, который относится к группе элементов управления ActiveX, то после щелчка на нем вызывается связанная процедура обработки событий (например, `CommandButton1_Click`), которая находится в модуле кода объекта `Лист` (Sheet). Связать макрос с этой процедурой нельзя.

Если Excel находится в режиме конструктора, тестирование элементов управления невозможно. В этом случае нужно выйти из режима конструктора, щелкнув на кнопке Разработчик⇨Элементы управления⇨Режим конструктора (Developer⇨Controls⇨Design mode). Эта кнопка работает, как переключатель.

Настройка элементов управления пользовательского диалогового окна

После того как элемент управления будет помещен в диалоговое окно, можно его переместить и изменить его размер. Воспользуйтесь стандартными методиками управления графическими объектами с помощью мыши.

Совет

Можно выделить несколько элементов управления. Для этого следует удерживать нажатой клавишу <Shift> и щелкать на объектах либо обвести указателем мыши необходимые элементы управления.

В форме `UserForm` содержатся вертикальные и горизонтальные направляющие, которые помогают выровнять добавленные в диалоговое окно элементы управления. При добавлении или перемещении элемент управления привязывается к направляющим, что облегчает упорядочение таких элементов в окне. Если вы не используете направляющие, можете их отключить, выбрав в VBE команду Tools⇨Options (Сервис⇨Параметры). В диалоговом окне Options (Параметры) перейдите на вкладку General (Общие) и выберите соответствующие параметры в разделе Form Grid Settings (Настройка направляющих формы).

Меню Format (Формат) окна VBE предоставляет несколько команд, которые позволяют точно разместить и выровнять элементы управления в диалоговом окне. Перед использованием этих команд необходимо указать элементы управления, к которым они будут применяться. Эти команды выполняют свои задачи так, как и ожидается. На рис. 13.4 показано диалоговое окно с несколькими элементами управления `OptionButton` в процессе выравнивания. На рис. 13.5 показано окно с теми же элементами управления, которые выровнены по вертикали и горизонтали.

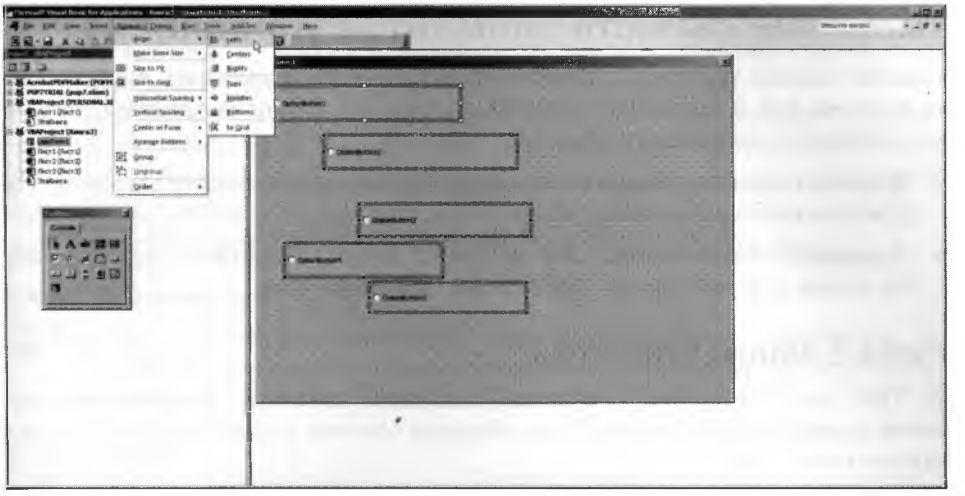


Рис. 13.4. Воспользуйтесь командой Format⇨Align для выравнивания элементов управления

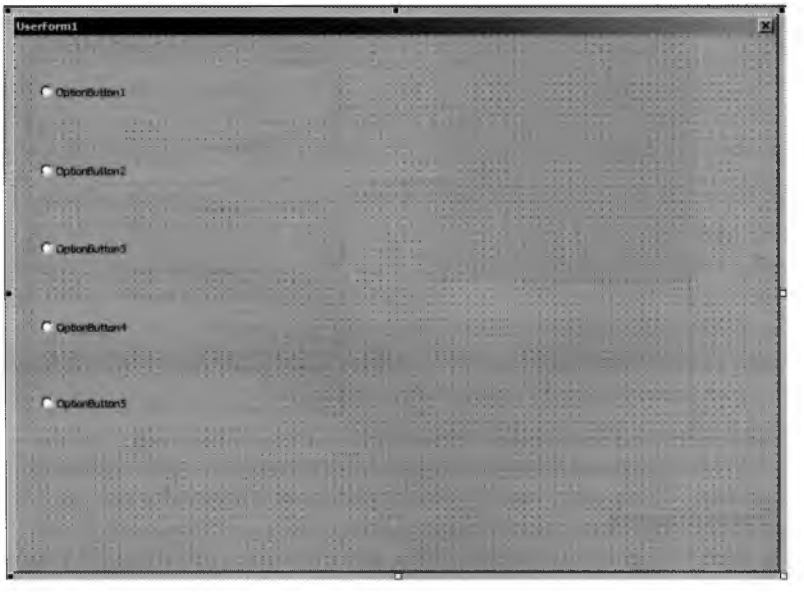


Рис. 13.5. Так выглядят элементы управления `OptionButton` после завершения выравнивания

Совет

Если выделено несколько элементов управления, на выделяющей рамке последнего из них вместо обычных черных маркеров появляются белые. Это означает, что последний элемент управления играет роль основы, на базе которой определяются размеры и расположение других элементов управления.

Изменение свойств элементов управления

Каждый элемент управления характеризуется набором параметров, которые определяют внешний вид и поведение элемента управления. Свойства элемента управления можно изменять в следующих случаях.

- **В момент проектирования** при разработке пользовательского диалогового окна. Для этого используется окно Properties (Свойства).
- **В процессе выполнения**, когда пользовательское диалоговое окно отображается на экране. Для этого воспользуйтесь инструкциями VBA.

Работа с окном Properties

В VBE окно Properties (Свойства) позволяет изменять свойства выделенного элемента управления (это может быть обычный элемент управления или сама форма `UserForm`) (рис. 13.6).

Рис. 13.6. Окно Properties для выделенного элемента управления `OptionButton`

Примечание

В окне Properties (Свойства) есть две вкладки. На вкладке Alphabetic (По алфавиту) свойства выбранного объекта отображаются в алфавитном порядке. На вкладке Categorized (По категориям) эти свойства сгруппированы по категориям. Обе вкладки отображают одни и те же свойства, но по-разному.

Для того чтобы изменить свойство, необходимо щелкнуть на нем и ввести новое значение. Некоторые свойства могут принимать только ограниченный набор допустимых значений, выбираемых из соответствующего списка. После щелчка на таком свойстве в окне Properties будет отображена кнопка со стрелкой, указывающей вниз. Щелкните на этой кнопке, чтобы выбрать значение из предложенного списка. Например, свойство `TextAlign` может принимать одно из следующих значений: `1` — `fmTextAlignLeft`, `2` — `fmTextAlignCenter` и `3` — `fmTextAlignRight`.

После выделения отдельных свойств (например, Font и Picture) рядом с ними отображается небольшая кнопка с троеточием. Щелчок на этой кнопке приводит к вызову диалогового окна настройки свойства.

Свойство Picture элемента управления Image стоит рассмотреть отдельно, поскольку для него необходимо указать графический файл. Еще один вариант — вставить изображение из буфера обмена. В последнем случае следует сначала скопировать его в буфер обмена, а затем выбрать свойство Picture элемента управления Image и нажать комбинацию клавиш <Ctrl+V> для вставки содержимого буфера обмена.

Примечание

Если выделить два или более элементов управления одновременно, в окне Properties отобразятся только те свойства, которые являются общими для этих объектов.

Совет

Объекту UserForm присущ ряд свойств, значения которых можно изменять. Эти свойства применяются в качестве значений, заданных по умолчанию, для элементов управления, которые добавляются в пользовательские диалоговые окна. Например, если изменить свойство Font пользовательского диалогового окна, все добавленные в окно элементы управления будут применять этот шрифт.

Общие свойства

Каждый элемент управления имеет как собственный набор уникальных свойств, так и ряд общих свойств, присущих другим элементам управления. Например, все элементы управления имеют свойство Name и свойства, определяющие его размер и расположение на форме (Height, Width, Left и Right).

Если вы собираетесь работать с элементом управления с помощью кода VBA, присвойте ему значащее имя. Например, первый элемент управления OptionButton, который добавлен в пользовательское диалоговое окно, по умолчанию получит имя OptionButton1. В коде ссылка на этот объект будет выглядеть следующим образом:

```
OptionButton1.Value = True
```

Но если элементу управления OptionButton присвоить описательное имя (например, obLandscape), то можно использовать такой оператор:

```
obLandscape.Value = True
```

Совет

Многие пользователи предпочитают имена, которые указывают на тип объекта. В предыдущем примере был использован префикс ob, который указывает на то, что объект является элементом управления OptionButton.

Можно изменять значения свойств нескольких элементов управления одновременно. Например, вы вправе создать на форме несколько элементов управления OptionButton и выровнять их по левому краю. Для этого достаточно выделить все элементы управления OptionButton и изменить значение свойства Left в окне Properties. Все выделенные элементы управления примут новое значение свойства Left.

Диалоговое справочное руководство является наилучшим способом получения информации о свойствах различных элементов управления. Щелкните на свойстве в окне Properties и нажмите клавишу <F1>. На рис. 13.7 показан пример справочных сведений, приведенных для выделенного свойства.

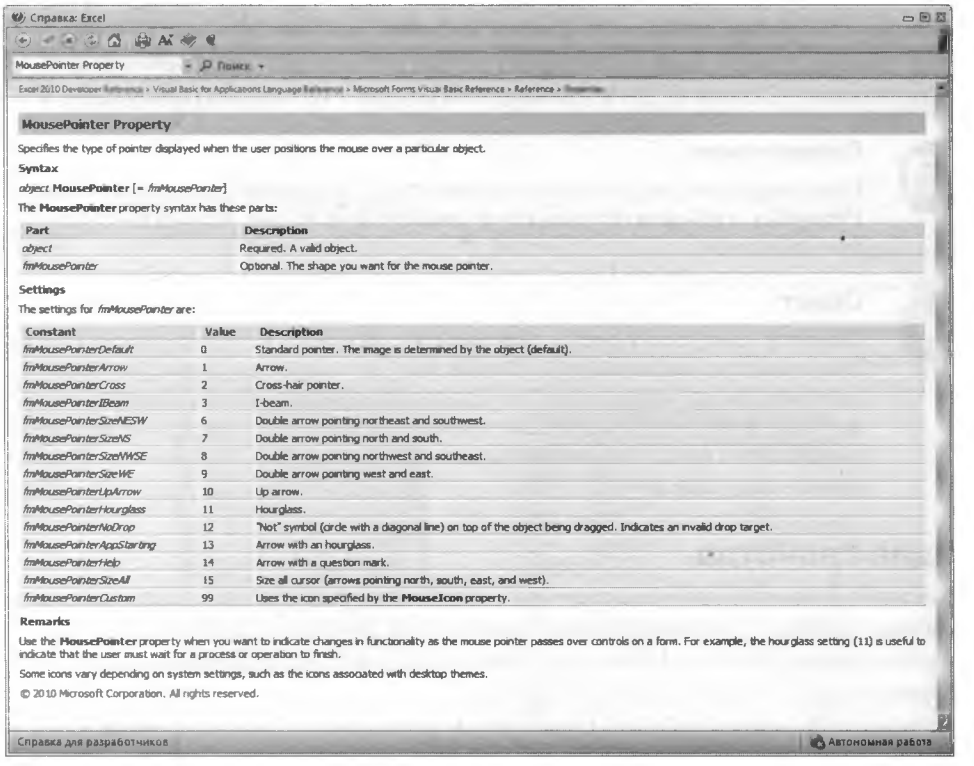


Рис. 13.7. В справочной системе можно найти сведения о каждом свойстве любого элемента управления

Советы по использованию клавиатуры

Многие пользователи предпочитают перемещаться по диалоговым окнам с помощью клавиатуры. Комбинации клавиш <Tab> и <Shift+Tab> позволяют циклически переключаться между элементами управления. Чтобы удостовериться, что диалоговое окно корректно реагирует на команды с клавиатуры, обратите внимание на такие моменты: порядок просмотра элементов управления и комбинации клавиш.

Изменение порядка просмотра (активизации)

Порядок просмотра определяет последовательность, в которой активизируются элементы управления после нажатия пользователем клавиши <Tab> или комбинации клавиш <Shift+Tab>. Кроме того, порядок активизации указывает, какой элемент управления по умолчанию выделяется на форме первым. Если пользователь вводит текст в элемент управления `TextBox`, то этот элемент считается активным. Если после этого щелкнуть на элементе управления `OptionButton`, то именно он станет активным. Элемент управления, назначенный первым для просмотра, будет активным в момент открытия диалогового окна.

Для того чтобы указать порядок активизации, выберите команду View⇨Tab Order (Вид⇨Порядок просмотра). Кроме того, можно щелкнуть правой кнопкой мыши на диалоговом окне и выбрать пункт Tab Order из появившегося контекстного меню. В любом случае Excel отобразит диалоговое окно Tab Order (Порядок просмотра), которое показано на рис. 13.8. В диалоговом окне Tab Order содержится упорядоченный список всех элементов управления в последовательности, которая соответствует порядку активизации объектов в пользовательском диалоговом окне. Чтобы переместить элемент управления в списке, выберите его и щелкните на кнопке Move Up (Переместить вверх) или Move Down (Переместить вниз). Можно одновременно перемещать более одного элемента управления (при выделении элементов удерживайте клавишу <Shift> или <Ctrl>).

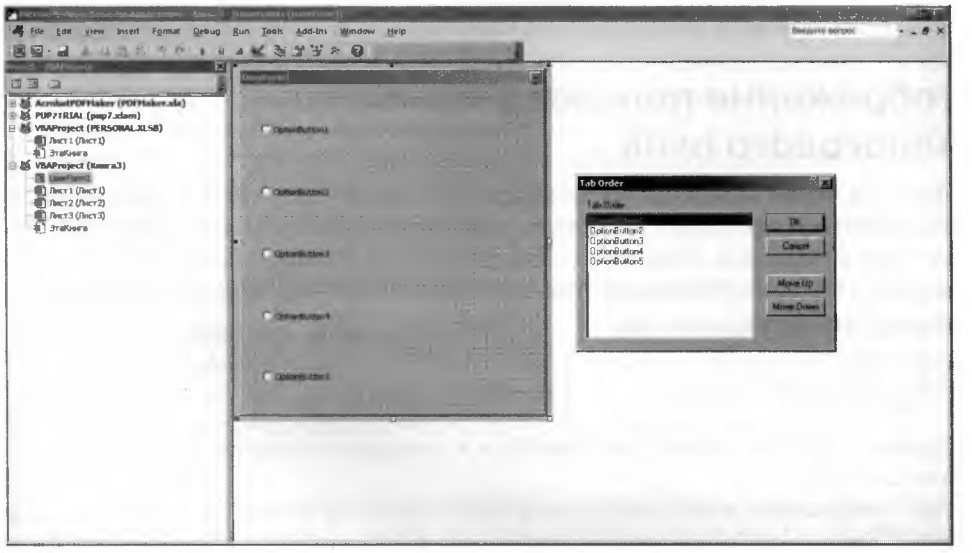


Рис. 13.8. В диалоговом окне Tab Order изменяется порядок просмотра элементов управления

С другой стороны, можно указать порядок активизации элемента управления с помощью окна Properties. Первый активизируемый элемент управления будет иметь свойство `TabIndex`, установленное равным `0`. Изменение значения свойства `TabIndex` текущего объекта приведет к изменению значений свойств `TabIndex` других элементов управления. Изменения вносятся автоматически. Вы можете удостовериться в том, что значения свойства `TabIndex` всех элементов управления не больше количества элементов управления в диалоговом окне. Если нужно удалить элемент управления из списка активизируемых объектов, то присвойте его свойству `TabStop` значение `False`.

Примечание

Одни элементы управления, такие как `Frame` и `MultiPage`, служат контейнерами для других элементов управления. Элементы управления в таком контейнере имеют собственный порядок просмотра (активизации). Для установки порядка просмотра группы элементов управления `OptionButtons`, находящихся внутри элемента управления `Frame`, выделите элемент управления `Frame` до того, как будет выполнена команда View⇨Tab Order.

Назначение комбинаций клавиш

Большинству элементов управления диалогового окна можно назначить комбинацию клавиш. Таким образом, пользователь получит доступ к элементу управления, нажав <Alt> и указанную клавишу. Применив свойство `Accelerator` в окне Properties, можно определить клавишу для активизации элемента управления.

Совет

Некоторые элементы управления, например `TextBox`, лишены свойства `Accelerator`, поскольку не отображают значение свойства `Caption`. Но к таким элементам можно получить доступ с помощью клавиатуры, воспользовавшись свойством `Label`. Присвойте клавишу элементу управления `Label` и расположите его в порядке просмотра перед элементом `TextBox`.

Отображение пользовательского диалогового окна

Для того чтобы отобразить пользовательское диалоговое окно с помощью VBA, необходимо создать процедуру, которая вызывает метод `Show` объекта `UserForm`. Форму `UserForm` невозможно отобразить, не выполнив как минимум одну строку кода VBA. Если объект `UserForm` называется `UserForm1`, то следующая процедура отобразит это пользовательское диалоговое окно.

```
Sub ShowForm()
   UserForm1.Show
End Sub
```

Данная процедура должна располагаться в стандартном модуле VBA, а не в модуле формы `UserForm`.

При отображении пользовательская форма остается на экране до тех пор, пока ее не скроют. Обычно в нее добавляют элемент управления `CommandButton`, который запускает процедуру закрытия формы. Эта процедура либо выгружает пользовательскую форму с помощью метода `Unload`, либо удаляет ее с экрана с помощью метода `Hide` объекта `UserForm`. Детально с каждым из них вы познакомитесь в приведенных далее примерах.

Отображение немодальной формы

По умолчанию отображается модальная форма. Это означает, что форма должна исчезнуть с экрана прежде, чем пользователь выполнит какие-либо действия на рабочем листе (т.е. редактирование данных невозможно). Немодальную форму также можно отобразить. В этом случае вы вправе продолжать работу в Excel, не скрывая саму форму. Для отображения немодальной формы используется следующий синтаксис:

```
UserForm1.Show vbModeless
```

Тестирование пользовательского диалогового окна

Обычно в процессе разработки возникает необходимость в тестировании формы `UserForm`. Для этого можно воспользоваться одним из следующих трех способов:

- выполните команду Run⇨Run Sub/UserForm (Выполнить⇨Выполнить процедуру/пользовательскую форму);

- нажмите клавишу <F5>;
- щелкните на кнопке Run Sub/UserForm, которая находится на стандартной панели инструментов.

Выбор одного из этих методов приводит к запуску события инициализации диалогового окна. Как только диалоговое окно будет отображено в тестовом режиме, можно проверить порядок активизации объектов, а также поддержку комбинаций клавиш.

Отображение пользовательского диалогового окна на основе значения переменной

В некоторых случаях приходится выбирать, какое окно `UserForm` будет отображено. Если название пользовательского диалогового окна хранится в виде строковой переменной, можно воспользоваться методом `Add` для добавления объекта `UserForm` в коллекцию `UserForms` с последующим обращением к методу `Show` из коллекции `UserForms`. В приведенном ниже примере название объекта `UserForm` присваивается переменной `MyForm`, после чего отображается пользовательское диалоговое окно.

```
MyForm = "UserForm1"
UserForms.Add(MyForm).Show
```

Загрузка пользовательского диалогового окна

В VBA поддерживается оператор `Load`. Загрузка пользовательского диалогового окна приводит к сохранению объекта формы в памяти. Однако до тех пор пока не будет выполнен метод `Show`, форма останется невидимой для остальной части программы. Для загрузки окна `UserForm1` воспользуйтесь следующим оператором:

```
Load UserForm1
```

Если вы применяете сложное диалоговое окно, то вам может понадобиться предварительно загрузить его в память, чтобы в случае необходимости быстро отобразить его с помощью метода `Show`. Как правило, использовать метод `Load` в подобном случае не имеет смысла.

О процедурах обработки событий

Как только диалоговое окно появляется на экране, пользователь начинает с ним взаимодействовать — выбирать опции в элементе управления `ListBox`, щелкать на кнопках `CommandButton` и т.д. Используя официальную терминологию, можно сказать, что пользователь генерирует события. Например, щелчок на элементе управления `CommandButton` приводит к возникновению события `Click` объекта `CommandButton`. Вам необходимо создать процедуры, которые будут выполняться при возникновении соответствующих событий. Первые называются обработчиками событий.

Примечание

Процедуры обработки событий находятся в модуле кода объекта `UserForm`. Наряду с этим процедура обработки события может вызывать другие процедуры, которые находятся в стандартном модуле VBA.

В коде VBA можно изменять свойства элементов управления, пока пользовательское диалоговое окно отображается на экране (т.е. на этапе выполнения). Например, можно

назначить элементу управления ListBox процедуру, которая изменяет текст элемента управления Label при выборе элемента списка. Подобное поведение элементов управления подробно рассмотрено далее.

Закрытие пользовательского диалогового окна

Для закрытия формы UserForm1 воспользуйтесь командой Unload, как показано ниже:

```
Unload UserForm1
```

Если же код находится в модуле кода формы UserForm, воспользуйтесь следующим оператором:

```
Unload Me
```

В этом случае ключевое слово Me применяется для идентификации пользовательской формы.

Обычно в коде VBA команда Unload выполняется только после того, как форма UserForm выполнит все свои функции. Например, форма UserForm может содержать элемент управления CommandButton, который используется в качестве кнопки OK. Щелчок на этой кнопке приводит к выполнению заранее определенного макроса. Одна из функций макроса заключается в выгрузке формы UserForm из памяти. В результате пользовательское диалоговое окно отображается на экране до тех пор, пока макрос, содержащий оператор Unload, не завершает свою работу.

Когда форма UserForm выгружается из памяти, элементы управления, содержавшиеся на ней, возвращаются в первоначальное состояние. Другими словами, в коде нельзя обращаться к значениям, указываемым пользователем, после того как форма будет выгружена из памяти. Если значения, введенные пользователем, будут применяться позже (после выгрузки диалогового окна UserForm), то необходимо присвоить их переменной с областью действия Public, которая определена в стандартном модуле VBA. Кроме того, значение всегда можно сохранить в ячейке листа.

Примечание

Окно формы UserForm автоматически выгружается из памяти после того, как пользователь щелкает на кнопке Закрыть (Close) (обозначается символом X в заголовке окна). Это действие также приводит к возникновению события QueryClose объекта UserForm, за которым следует событие Terminate пользовательского диалогового окна.

Объект UserForm может использовать метод Hide. После его вызова диалоговое окно исчезает, но остается в памяти, поэтому в коде можно получить доступ к различным свойствам элементов управления. Ниже приведен пример оператора, который скрывает диалоговое окно.

```
UserForm1.Hide
```

Также можно воспользоваться следующим оператором (если ваш код находится в модуле кода объекта UserForm):

```
Me.Hide
```

Если по какой-либо причине пользовательское диалоговое окно должно быть немедленно скрыто в процессе выполнения макроса, воспользуйтесь методом Hide в самом начале процедуры, а затем укажите команду DoEvents. Например, в следующей проце-

уре форма UserForm немедленно исчезнет после того, как пользователь щелкнет на нопке CommandButton1. Последний оператор процедуры выгружает пользовательское диалоговое окно из памяти.

```
rivate Sub CommandButton1_Click()
  Me.Hide
  Application.ScreenUpdating = True
  For r = 1 To 10000
  Cells(r, 1) = r
  Next r
  Unload Me
nd Sub
```

В рассматриваемом примере переменной ScreenUpdating присвоено значение rue, в результате чего Excel полностью скрывает окно UserForm. Если этот оператор е использовать, окно UserForm остается видимым.

Перекрестная ссылка

В главе 15 будет показано, как отображать индикатор хода выполнения макроса. При этом используется тот факт, что диалоговое окно остается на экране до тех пор, пока выполняется макрос.

Пример создания пользовательского диалогового окна

Если вы еще никогда не создавали пользовательские диалоговые окна, то обратите внимание на пример, приведенный в этой главе. В нем вы найдете пошаговые инструкции по созданию простого диалогового окна и разработке процедуры VBA для поддержки этого диалогового окна.

В приведенном примере представлено диалоговое окно, предназначенное для получения имени и пола пользователя. В диалоговом окне вы найдете элемент управления TextBox, используемый для ввода имени, и три элемента управления OptionButton для указания пола ("мужчина", "женщина" и "не определен"). Информация, полученная в диалоговом окне, заносится в пустую строку рабочего листа.

Создание пользовательской формы

На рис. 13.9 показано завершенное пользовательское диалоговое окно, созданное в этом примере. Для получения наилучших результатов начните с новой рабочей книги, содержащей только один рабочий лист. Выполните следующие действия.

1. Нажмите комбинацию клавиш <Alt+F11> для активизации VBE.
2. В окне Project (Проект) выберите проект рабочей книги и выполните команду Insert⇨UserForm (Вставка⇨Форма UserForm) для добавления пустой формы UserForm.
3. Свойству Caption формы UserForm присвоено заданное по умолчанию значение UserForm1. Воспользуйтесь окном Properties (Свойства) для изменения значения свойства Caption формы UserForm на Укажите имя и пол. (Если окно Properties не отображается, нажмите клавишу <F4>.)

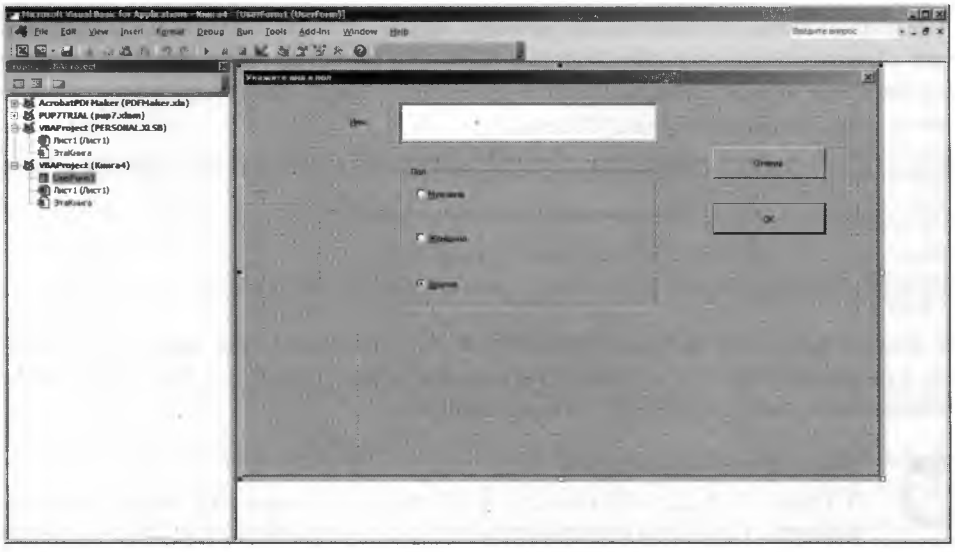


Рис. 13.9. В этом диалоговом окне пользователю предлагается ввести имя и указать пол

4. Добавьте элемент управления `Label` и настройте его свойства, как указано ниже.

Свойство	Значение
Accelerator	И
Caption	Имя:
TabIndex	0

5. Добавьте элемент управления `TextBox` и измените его свойства следующим образом.

Свойство	Значение
Name	TextName
TabIndex	1

6. Добавьте элемент управления `Frame` и измените его свойства следующим образом.

Свойство	Значение
Caption	Пол
TabIndex	2

7. Добавьте элемент управления `OptionButton` в состав элемента `Frame` и измените его свойства следующим образом.

Свойство	Значение
Accelerator	М
Caption	Мужчина
Name	OptionMale
TabIndex	0

8. Добавьте еще один элемент управления `OptionButton` внутри элемента `Frame` и измените его свойства следующим образом.

Свойство	Значение
Accelerator	Ж
Caption	Женщина
Name	OptionFemale
TabIndex	1

9. Добавьте еще один элемент управления `OptionButton` внутри элемента `Frame` и измените его свойства следующим образом.

Свойство	Значение
Accelerator	Д
Caption	Другое
Name	OptionUnknown
TabIndex	2
Value	True

10. Добавьте элемент управления `CommandButton` за пределами элемента `Frame` и измените его свойства следующим образом.

Свойство	Значение
Caption	OK
Default	True
Name	OKButton
TabIndex	3

11. Добавьте еще один элемент управления `CommandButton` и настройте его свойства следующим образом.

Свойство	Значение
Caption	Отмена
Default	False
Name	CloseKButton
TabIndex	4

Совет

При создании нескольких похожих элементов управления может оказаться, что быстрее копировать существующий элемент управления, чем создавать новый. Для копирования элемента управления удерживайте нажатой клавишу <Ctrl> при перетаскивании элемента управления, что приведет к созданию копии. После этого останется изменить свойства скопированного элемента управления.

Создание кода для отображения диалогового окна

После создания элементов управления на лист необходимо добавить элемент управления ActiveX, называемый `Кнопка` (CommandButton). Эта кнопка будет запускать процедуру, которая предназначена для отображения формы `UserForm`. Для этого выполните следующие действия.

1. Перейдите в окно Excel (воспользуйтесь комбинацией клавиш <Alt+F11>).
2. Выберите команду **Разработчик⇨Элементы управления⇨Вставить** (Developer⇨Controls⇨Insert) и щелкните на значке **Кнопка** (CommandButton), который находится в разделе **Элементы ActiveX** (ActiveX Controls).

3. Перетащите кнопку на рабочий лист. При необходимости измените подпись объекта `Кнопка`. Для этого щелкните на нем правой кнопкой мыши и выберите в контекстном меню команду **Объект CommandButton⇨Edit** (Объект Кнопка⇨Правка). После этого измените текст, который отображается на кнопке. Для изменения других свойств объекта щелкните на нем правой кнопкой мыши и выберите команду **Properties** (Свойства). Выполните необходимые изменения в диалоговом окне **Properties**.
4. Дважды щелкните на объекте `CommandButton`. Это приведет к активизации VBE. При этом отобразится модуль кода для листа с открытой пустой процедурой обработки событий объекта `CommandButton`, который расположен на рабочем листе.
5. Введите единственный оператор в процедуру `CommandButton1_Click`, как показано на рис. 13.10. В этой краткой процедуре используется метод `Show` объекта `UserForm1` для отображения пользовательского диалогового окна.

Рис. 13.10. Процедура `CommandButton1_Click` вызывается после щелчка на кнопке рабочего листа

Тестирование диалогового окна

Следующий этап заключается в проверке работы процедуры, отображающей диалоговое окно.

Примечание

После щелчка на кнопке, находящейся на рабочем листе, ничего не произойдет. Точнее, кнопка будет выделена, но это не приведет к инициализации каких-либо действий. Причина этого заключается в том, что программа Excel по-прежнему остается в режиме конструктора, в который она переходит автоматически после добавления элемента управления ActiveX. Для выхода из режима конструктора щелкните на кнопке **Разработчик⇨Элементы управления⇨Режим конструктора** (**Developer⇨Controls⇨Design Mode**). Если же требуется изменить кнопку, снова перейдите в режим конструктора.

После выхода из режима конструктора щелчок на кнопке приведет к отображению пользовательского диалогового окна (рис. 13.11).

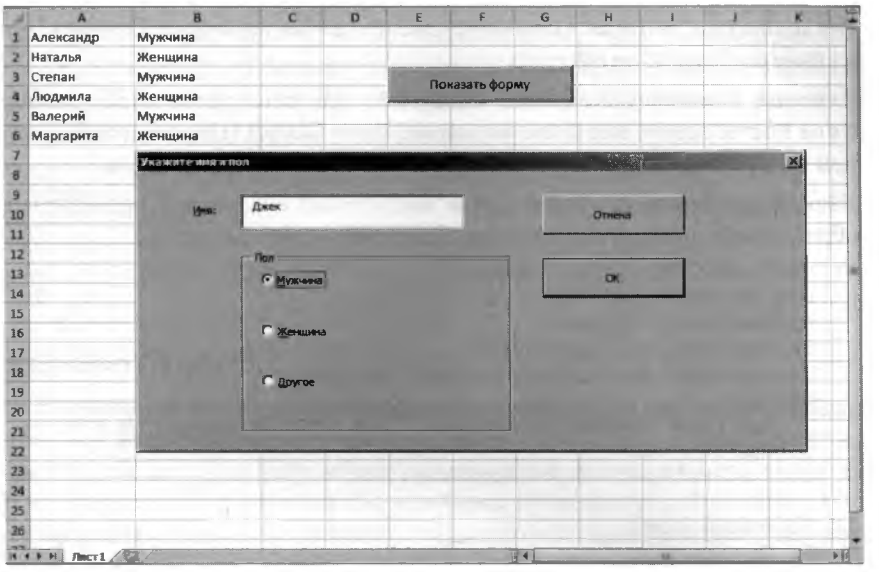


Рис. 13.11. Процедура обработки события `Click` объекта `CommandButton` отображает форму `UserForm`

Когда диалоговое окно будет отображено, введите произвольный текст в текстовом поле и щелкните на кнопке OK. В результате ничего не произойдет, что совершенно естественно, так как для объекта `UserForm` не создано ни одной процедуры обработки событий.

Примечание

Для закрытия диалогового окна щелкните на крестике в его заголовке.

Добавление процедур обработки событий

В этом разделе рассматривается создание процедур, которые обрабатывают события, возникающие после открытия пользовательского диалогового окна. Выполните следующие действия.

1. Для активизации VBE нажмите комбинацию клавиш <Alt+F11>.
2. Удостоверьтесь в том, что пользовательское окно отображено на экране, и дважды щелкните на кнопке Отмена.
3. После этого активизируется окно кода для формы `UserForm`, а также добавляется пустая процедура `CloseButton_Click`. Обратите внимание, что название процедуры состоит из имени объекта, символа подчеркивания и названия обрабатываемого события.
4. Измените процедуру, как указано ниже. (Это обработчик события `Click` объекта `CloseButton`.)

```
Private Sub CloseButton_Click()
   Unload UserForm1
End Sub
```

Эта процедура, которая вызывается после щелчка на кнопке Отмена, выгружает из памяти форму UserForm1.

5. Нажмите комбинацию клавиш <Shift+F7> для повторного отображения формы UserForm1 (либо щелкните на значке View Object (Просмотр объекта) в верхней части окна Project Explorer (Просмотр объектов)).
6. Дважды щелкните на кнопке OK и введите код следующей процедуры (это код обработчика событий для события Click объекта OKButton).

```
Private Sub OKButton_Click()
   Dim NextRow As Long
'  Активизация листа
   Sheets("Лист1").Activate
'  Определение следующей пустой строки
   NextRow = _
      Application.WorksheetFunction.CountA(Range("A:A")) + 1
'  Передача имени
   Cells(NextRow, 1) = TextName.Text

'  Передача пола
   If OptionMale Then Cells(NextRow, 2) = "Мужчина"
   If OptionFemale Then Cells(NextRow, 2) = "Женщина"
   If OptionUnknown Then Cells(NextRow, 2) = "Другое"

'  Очистка элементов управления для следующих записей
   TextName.Text = ""
   OptionUnknown = True
   TextName.SetFocus
End Sub
```

7. Перейдите в окно Excel и щелкните на кнопке еще раз, чтобы отобразить пользовательское диалоговое окно. Запустите процедуру повторно.

   Элементы управления диалогового окна должны функционировать правильно. Теперь с помощью пользовательской формы можно добавлять новые имена в рабочий лист.

Процедура OKButton_Click работает следующим образом. Сначала она проверяет, активен ли лист Лист1. После этого запускается функция Excel СЧЁТЗ (COUNTA) для определения следующей пустой ячейки в столбце A. Затем текст из текстового поля TextBox передается в определенную ячейку столбца A. С помощью операторов If определяется выделенный элемент управления OptionButton, что обеспечивает запись соответствующего текста в столбец B (Мужчина, Женщина, Другое). Наконец, диалоговое окно перезапускается (чтобы обеспечить возможность введения следующей записи). Заметим, что щелчок на кнопке OK не приведет к закрытию диалогового окна. Для завершения ввода данных (и выгрузки пользовательского диалогового окна) щелкните на кнопке Отмена.

Проверка правильности введенных данных

Приведенному в этом разделе примеру следует уделить особое внимание. Возможно, вы заметили, что не устранена небольшая проблема — отсутствует проверка введенных

в текстовом поле данных (вы не знаете, ввел ли пользователь свое имя). Следующий код добавлен в процедуру OKButton_Click перед оператором вставки текста на рабочий лист. Он проверяет, ввел ли пользователь свое имя (на самом деле проверяется наличие любого текста) в поле TextBox. Если текстовое поле TextBox осталось пустым, то выводится соответствующее сообщение, и текстовое поле снова становится активным. Таким образом, пользователь сможет приступить к введению своего имени. Оператор Exit Sub завершает процедуру без выполнения дополнительных действий.

```
' Проверка ввода имени
  If TextName.Text = "" Then
     MsgBox "Введите имя."
     TextName.SetFocus
     Exit Sub
  End If
```

Ура, заработало!

После внесения соответствующих исправлений диалоговое окно будет работать безупречно (не забудьте проверить работоспособность комбинаций клавиш). На практике вам может потребоваться собрать много дополнительной информации, а не только сведения об имени и поле пользователя. Но в любом случае следует применять изложенные выше принципы. Вам останется добавить в диалоговое окно большее количество элементов управления.

Компакт-диск

Рабочая книга с рассмотренным выше примером находится на прилагаемом компакт-диске в файле get name and sex.xlsm.

События объекта UserForm

Каждый элемент управления в форме UserForm (а также сам объект UserForm) разрабатывается для того, чтобы реагировать на определенные события. Эти события возникают в результате действий пользователя или генерируются программой Excel. Например, щелчок на кнопке (CommandButton) приводит к возникновению события Click объекта CommandButton. Можно создать код, который будет выполняться при возникновении определенного события.

Некоторые действия приводят к возникновению сразу нескольких событий. Например, щелчок на кнопке со стрелкой, направленной вверх, в элементе управления SpinButton приведет к возникновению события SpinUp и события Change. После того как пользовательское диалоговое окно будет отображено с помощью метода Show, Excel сгенерирует события Initialize и Activate объекта UserForm. (На самом деле событие Initialize генерируется после загрузки объекта UserForm в память и до его фактического отображения.)

Перекрестная ссылка

В Excel также поддерживаются события, связанные с объектами Sheet (Лист), Chart (Диаграмма) и ThisWorkbook (ЭтаКнига). Эти события будут подробно рассмотрены в главе 18.

Получение дополнительных сведений о событиях

Для того чтобы получить информацию о событиях, которые генерируются конкретным элементом управления, выполните следующие действия.

1. Добавьте элемент управления в пользовательское диалоговое окно.
2. Дважды щелкните на элементе управления, чтобы открыть модуль кода для объекта `UserForm`. VBE вставит пустую процедуру обработки события, принятого по умолчанию.
3. Щелкните на раскрывающемся списке в правом верхнем углу окна модуля и просмотрите полный список событий, которые поддерживаются текущим элементом управления.

   На рис. 13.12 показан список событий для элемента управления `CheckBox`.
4. Выберите событие из списка, и VBE создаст пустой обработчик события.

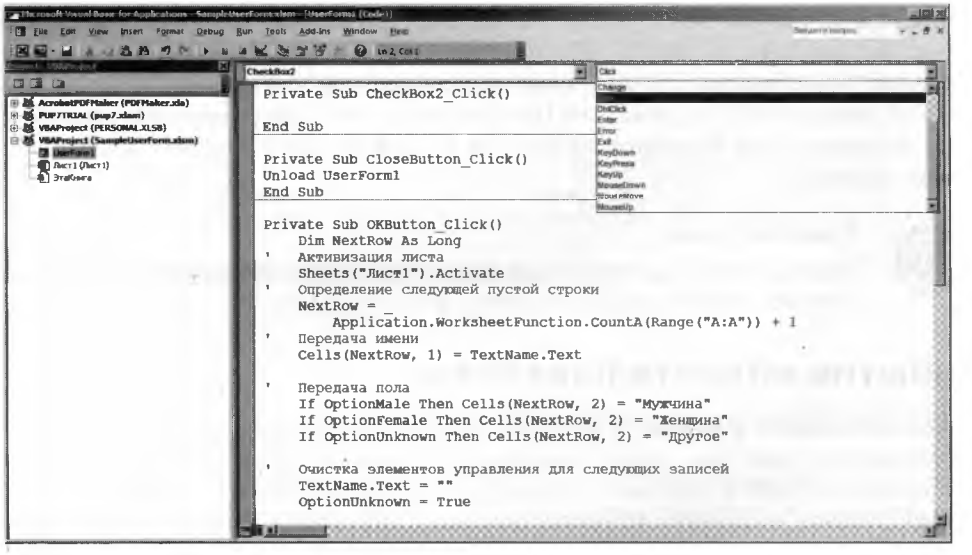


Рис. 13.12. Список событий для элемента управления `CheckBox`

Примечание

Для получения дополнительных сведений о событии обратитесь к интерактивной справочной системе. Здесь же можно найти описание событий, поддерживаемых тем или иным элементом управления. После нахождения события для объекта убедитесь в том, что отображается содержание справочной системы. Здесь можно увидеть список всех других событий для объекта.

Предупреждение

Имя процедуры обработки событий включает имя объекта, который сгенерировал событие. Таким образом, если изменить имя элемента управления, придется внести соответствующие изменения и в имя процедуры обработки события. Имя процедуры не изменяется автоматически! Чтобы облегчить работу, присвойте описательные имена элементам управления до того, как приступите к созданию процедуры обработки соответствующих событий.

События объекта UserForm

Несколько событий непосредственно связано с отображением и выгрузкой объекта UserForm.

- Initialize. Происходит перед загрузкой и отображением формы UserForm. Не происходит, если объект UserForm до этого был скрыт.
- Activate. Происходит в момент активизации объекта UserForm.
- Deactivate. Происходит в момент деактивизации объекта UserForm. Не происходит при сокрытии формы UserForm.
- QueryClose. Происходит перед выгрузкой объекта UserForm.
- Terminate. Происходит после выгрузки объекта UserForm.

Примечание

Важно правильно выбрать подходящее событие для процедуры обработки событий, а также проанализировать порядок выполнения событий. Использование метода Show приводит к возникновению событий Initialize и Activate (в указанном порядке). Применение команды Load приводит к вызову события Initialize. Команда Unload вызывает события QueryClose и Terminate (в указанном порядке). Метод Hide не приводит к вызову каких-либо событий.

Компакт-диск

На прилагаемом компакт-диске находится рабочая книга userform events.xlsm, которая управляет описанными событиями и отображает в момент возникновения события специальное сообщение. Если изучение событий объекта UserForm вызывает у вас затруднения, то, проанализировав код этого примера, вы получите ответы на многие вопросы.

События элемента управления SpinButton

Для того чтобы разобраться в концепции событий, в этом разделе мы подробно рассмотрим события, связанные с элементом управления SpinButton.

Компакт-диск

На прилагаемом компакт-диске содержится рабочая книга userform events.xlsm, которая демонстрирует применение событий, генерируемых объектами SpinButton и UserForm (первый содержится во втором). Рабочая книга включает несколько процедур обработки событий — по одной для каждого события элемента управления SpinButton и объекта UserForm. Каждая из этих процедур создает окно сообщения, которое указывает на возникновение соответствующего события.

В табл. 13.1 перечислены события элемента управления SpinButton.

Таблица 13.1. События элемента управления SpinButton

Событие	Описание
AfterUpdate	Происходит после того, как элемент управления изменяется с помощью пользовательского интерфейса
BeforeDragOver	Происходит в процессе выполнения операции перетаскивания объекта
BeforeDropOrPaste	Происходит перед тем, как пользователь отпустит перетаскиваемый объект или скопирует его из буфера обмена
BeforeUpdate	Происходит перед изменением элемента управления
Change	Происходит в момент изменения свойства Value
Enter	Происходит перед тем, как элемент управления SpinButton будет активизирован после другого элемента управления этой же формы UserForm
Error	Происходит в момент обнаружения элементом управления ошибки. При этом элемент управления не может передать сведения об ошибке вызывающей программе
Exit	Происходит непосредственно перед деактивизацией элемента управления с последующей активизацией другого элемента управления текущей формы
KeyDown	Происходит, когда пользователь нажимает клавишу, и активизируется объект
KeyPress	Происходит, когда пользователь нажимает клавишу на алфавитно-цифровой клавиатуре
KeyUp	Происходит, когда пользователь отпускает клавишу, а объект остается активным
SpinDown	Происходит, когда пользователь щелкает на нижней (или левой) кнопке элемента управления SpinButton
SpinUp	Происходит, когда пользователь щелкает на верхней (или правой) кнопке элемента управления SpinButton

Пользователь может управлять объектом SpinButton с помощью мыши или (если элемент управления активен) клавиш управления курсором.

События мыши

Когда пользователь щелкает мышью на верхней кнопке элемента управления SpinButton, происходят следующие события:

1) Enter (генерируется только в том случае, если элемент управления неактивен);
2) Change;
3) SpinUp.

События клавиатуры

Пользователь может нажать клавишу <Tab> для того, чтобы сделать активным элемент управления SpinButton. Только после этого можно использовать клавиши управления курсором для изменения значения элемента управления. Если все именно так и происходит, то события генерируются в следующем порядке:

1) Enter;
2) KeyDown;
3) Change;
4) SpinUp (или SpinDown);
5) KeyUp.

События, генерируемые кодом

Элемент управления SpinButton может изменяться в коде VBA, что также провоцирует возникновение соответствующих событий. Например, представленный далее оператор устанавливает свойство Value элемента управления SpinButton1 равным 0, а это приводит к возникновению события Change. Такой результат достигается только в том случае, если исходное свойство Value не равно нулю.

```
SpinButton1.Value = 0
```

Вы вправе предположить, что выполнить отмену генерирования событий можно, установив свойство EnableEvents объекта Application равным значению False. К сожалению, это свойство поддерживается только объектами, которые являются "истинными" в Excel: Workbook, Worksheet и Chart.

Совместное использование элементов управления SpinButton и TextBox

Элемент управления SpinButton имеет свойство Value, но не может отображать значение этого свойства. В большинстве случаев требуется, чтобы пользователь мог изменить значение элемента управления SpinButton непосредственно, а не многократно щелкая на элементе управления.

Эффективным решением может стать объединение элемента управления SpinButton с элементом управления TextBox, что позволяет пользователю вводить значение элемента управления SpinButton непосредственно, используя для этого поле элемента управления TextBox. Кроме того, щелчок на элементе управления SpinButton позволит изменить значение, отображаемое в элементе управления TextBox.

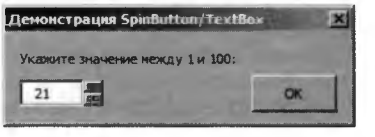


Рис. 13.13. Комбинирование элементов управления SpinButton и TextBox

На рис. 13.13 приведен простой пример. Свойство Min элемента управления SpinButton имеет значение 1, а свойство Max — значение 100. Таким образом, щелчок на одной из стрелок элемента управления SpinButton приведет к изменению значения в пределах от 1 до 100.

Компакт-диск

Рабочая книга, включающая рассматриваемый пример, доступна на прилагаемом компакт-диске в файле spinbutton and textbox.xlsm.

Код, реализующий "связывание" элементов управления SpinButton и TextBox, достаточно прост. В целом все сводится к созданию процедур обработки событий, которые будут синхронизировать свойство Value элемента управления SpinButton и свойство Text элемента управления TextBox.

Представленная далее процедура выполняется каждый раз при возникновении события Change элемента управления SpinButton. Таким образом, процедура выполняется тогда, когда пользователь щелкает на элементе управления SpinButton или изменяет его значение, нажав одну из клавиш управления курсором.

```
Private Sub SpinButton1_Change()
   TextBox1.Text = SpinButton1.Value
End Sub
```

Процедура просто приравнивает значение свойства Value элемента управления SpinButton к свойству Text элемента управления TextBox. В данном случае элементы управления имеют имена, заданные по умолчанию (SpinButton1 и TextBox1).

Если пользователь введет значение непосредственно в элемент управления TextBox, то будет сгенерировано событие Change, после чего должен выполняться следующий код.

```
Private Sub TextBox1_Change()
   NewVal = Val(TextBox1.Text)
   If NewVal >= SpinButton1.Min And _
      NewVal <= SpinButton1.Max Then _
      SpinButton1.Value = NewVal
End Sub
```

Эта процедура начинается с вызова функции VBA Val, которая преобразует текст элемента управления TextBox в числовое значение (если элемент управления TextBox содержит строку, то функция Val возвращает значение 0). Следующий оператор определяет, попадает ли значение в указанный диапазон допустимых значений. Если это так, то свойство Value элемента управления SpinButton устанавливается равным значению, которое введено в поле элемента управления TextBox.

Пример организован таким образом, что щелчок на кнопке OK (которая называется OKButton) передает значение элемента управления SpinButton в активную ячейку. Процедура обработки события Click элемента управления CommandButton выглядит следующим образом.

```
Private Sub OKButton_Click()
'   Введите значение в активную ячейку
   If CStr(SpinButton1.Value) = TextBox1.Text Then
      ActiveCell = SpinButton1.Value
      Unload Me
   Else
      MsgBox "Неправильное значение.", vbCritical
      TextBox1.SetFocus
      TextBox1.SelStart = 0
      TextBox1.SelLength = Len(TextBox1.Text)
   End If
End Sub
```

Данная процедура выполняет последнюю проверку: анализируются текст, введенный в поле элемента управления TextBox, и значения элемента управления SpinButton. Такая процедура обрабатывает ситуации неверного ввода данных. Например, если пользователь введет в поле элемента управления TextBox текст **3r**, то значение элемента управления SpinButton не изменится, а результат, который помещается в активную ячейку, будет отличным от ожидаемого. Обратите внимание, что значение свойства Value элемента управления SpinButton преобразуется в строку с помощью функции CStr. Это позволяет предотвратить ошибку, которая возникает, когда числовое значение сравнивается с текстовым. Если значение элемента управления SpinButton не соот-

ветствует содержимому элемента управления TextBox, то на экране отображается специальное сообщение. Причем объект TextBox активен, а его содержимое — выделено (с помощью свойств SelStart и SelLength). Таким образом, пользователю проще исправить неправильные значения.

О свойстве Tag

Каждый объект UserForm и каждый элемент управления имеет свойство Tag. Оно не представляет конечные данные и по умолчанию не имеет значения. Свойство Tag можно использовать для хранения информации, которая будет применена в программе.

Например, можно создать набор элементов управления TextBox в пользовательском диалоговом окне. От пользователя требуется ввести текст только в некоторые из них. В отдельные поля вводить текст необязательно. Можно применять свойство Tag для идентификации полей, которые нужно заполнять. В таком случае значение свойства Tag — это строка, например, Required. Поэтому при написании кода обработки введенных пользователем данных можно ссылаться на свойство Tag.

Приведенный ниже пример представляет собой функцию, которая проверяет все элементы управления TextBox объекта UserForm1 и возвращает количество пустых текстовых полей, которые "требуют" ввода информации.

```
Function EmptyCount()
   Dim ctl As Control
   EmptyCount= 0
   For Each ctl In UserForm1.Controls
      If TypeName(ctl) = "TextBox" Then
         If ctl.Tag = "Required" Then
            If ctl.Text = "" Then
               EmptyCount = EmptyCount + 1
            End If
         End If
      End If
   Next ctl
End Function
```

Работая с пользовательскими диалоговыми окнами, можно придумать другое достойное применение для свойства Tag.

Ссылка на элементы управления пользовательского диалогового окна

При работе с элементами управления, находящимися в форме UserForm, код VBA обычно содержится в модуле кода объекта UserForm. Кроме того, на элементы управления диалогового окна можно ссылаться из модуля кода VBA общего назначения. Для выполнения этой задачи необходимо задать правильную ссылку на элемент управления, указав имя объекта UserForm. В качестве примера рассмотрим процедуру, которая введена в модуле кода VBA. Эта процедура отображает пользовательское диалоговое окно, которое называется UserForm1.

```
Sub GetData()
   UserForm1.Show
End Sub
```

Предполагается, что в диалоговом окне UserForm1 содержится текстовое поле (TextBox1). Вам же необходимо указать значение текстового поля по умолчанию. Процедуру можно изменить следующим образом.

```
Sub GetData()
   UserForm1.TextBox1.Value = "Джон Доу"
   UserForm1.Show
End Sub
```

Еще одним способом установки значения по умолчанию является использование события Initialize объекта UserForm. Можно написать код процедуры UserForm_Initialize, который будет располагаться в модуле кода диалогового окна. Ниже представлен соответствующий пример.

```
Private Sub UserForm_Initialize()
   TextBox1.Value = "Джон Доу"
End Sub
```

Обратите внимание, что при обращении к элементу управления из модуля кода диалогового окна необязательно вводить в ссылку имя объекта UserForm. Подобное определение ссылок на элементы управления имеет свое преимущество: всегда можно воспользоваться средством Auto List Member, которое позволяет выбирать имена элементов управления из раскрывающегося списка.

Совет

Вместо того чтобы использовать фактическое имя объекта UserForm, предпочтительнее применить имя Me. В противном случае, если имя объекта UserForm изменится, вам не придется изменять все ссылки (с его участием) в коде.

Использование коллекций элементов управления

Элементы управления пользовательских диалоговых окон образуют отдельную коллекцию. Например, следующий оператор отображает количество элементов управления в форме UserForm1:

```
MsgBox UserForm1.Controls.Count
```

В VBA не поддерживаются коллекции для каждого типа элемента управления. Например, не существует коллекции элементов управления CommandButton. Но тип элемента управления можно определить с помощью функции TypeName. Следующая процедура использует структуру For Each для циклического просмотра элементов коллекции Controls. В результате отображается количество элементов управления CommandButton, которые входят в коллекцию элементов управления объекта UserForm1.

```
Sub CountButtons()
   Dim cbCount As Integer
   Dim ctl as Control
   cbCount = 0
   For Each ctl In UserForm1.Controls
      If TypeName(ctl) = "CommandButton" Then _
         cbCount = cbCount + 1
   Next ctl
   MsgBox cbCount
End Sub
```

Настройка панели инструментов Toolbox

Если объект UserForm активен в редакторе VBE, на панели Toolbox отображаются элементы управления, которые можно добавить в пользовательское диалоговое окно. В этом разделе рассматриваются способы настройки панели Toolbox.

Добавление новых страниц

Панель Toolbox изначально содержит одну вкладку. Щелкните на ней правой кнопкой мыши и в контекстном меню выберите параметр New Page (Добавить страницу), чтобы добавить новую вкладку на панель Toolbox. Кроме того, можно изменить текст, который отображается на вкладке. Для этого выберите параметр Rename (Переименовать) из контекстного меню.

Настройка или комбинирование элементов управления

Рекомендуется предварительно настроить элементы управления и сохранить их для дальнейшего использования. Можно, например, создать элемент управления CommandButton, который настроен на выполнение роли кнопки OK. Можно изменять следующие параметры кнопки: Width (Ширина), Height (Высота), Caption (Подпись), Default (По умолчанию) и Name (Имя). После этого перетащите модифицированный элемент управления CommandButton на панель инструментов Toolbox. Это приведет к созданию элемента управления. Щелкните на элементе управления правой кнопкой мыши, чтобы переименовать его или изменить значок.

Также можно создать раздел панели Toolbox, в котором будет содержаться несколько элементов управления. Например, вы вправе создать два элемента управления CommandButton, которые будут представлять кнопки OK и Отмена. Настройте их так, как это необходимо. Затем выберите обе кнопки и переместите их на панель инструментов Toolbox. Впоследствии можно использовать новый элемент управления панели Toolbox для быстрого создания необходимых кнопок.

Этот метод также применим к элементам управления, которые используются в качестве контейнера. Например, создайте элемент управления Frame и добавьте в него четыре модифицированных элемента управления OptionButton (соответствующим образом расположив их на форме). После этого перетащите элемент управления Frame на панель инструментов Toolbox, чтобы создать модифицированный элемент управления Frame.

Совет

Иногда нужно разместить модифицированные элементы управления на отдельной вкладке панели Toolbox. Таким образом, появляется возможность экспортировать вкладку панели Toolbox для совместного применения другими пользователями Excel. Для экспорта вкладки панели Toolbox щелкните на ней правой кнопкой мыши и выберите пункт меню Export Page (Экспорт вкладки).

Компакт-диск

На прилагаемом к книге компакт-диске находится страничный файл (под именем newcontrols.pag), который включает некоторые настроенные элементы управления. Можно импортировать этот файл в качестве новой вкладки окна Toolbox. Щелкните правой кнопкой мыши на вкладке и выберите команду Import Page (Импортировать вкладку). После этого укажите рас-

положение страничного файла. В результате панель Toolbox будет напоминать показанную на рис. 13.14. Новые элементы управления будут такими: изображение восклицательного знака, "критический" знак красной буквы x, изображение вопросительного знака, информационное изображение, кнопки OK и Отмена, элемент управления `Frame`, включающий четыре элемента управления `OptionButton`, элементы управления `TextBox` и `Spinner`, а также шесть элементов управления `CheckBox`.

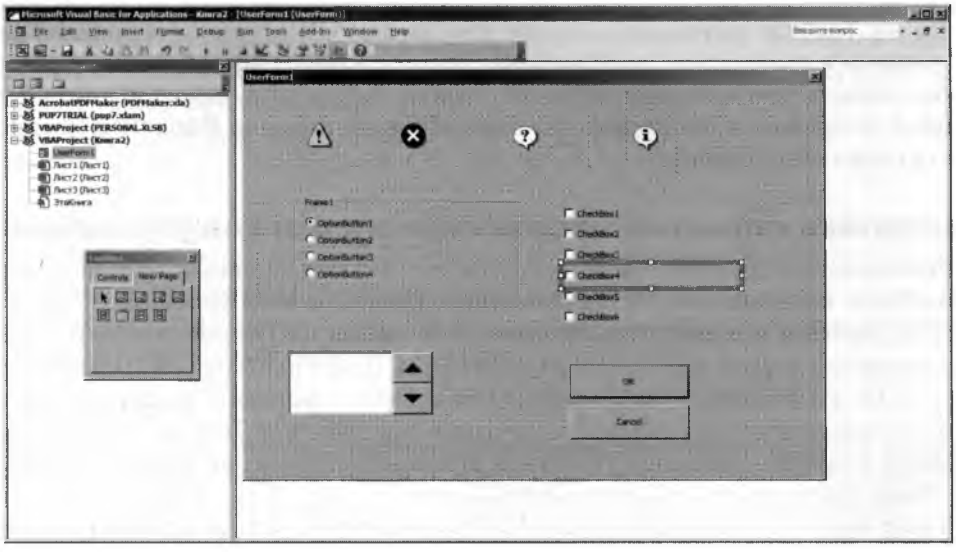


Рис. 13.14. В окне Toolbox появилась страница с новыми элементами управления

Добавление элементов управления ActiveX

В пользовательском диалоговом окне содержатся и другие элементы управления ActiveX, разработанные компанией Microsoft и независимыми производителями. Для того чтобы добавить дополнительные элементы управления ActiveX на панель инструментов Toolbox, щелкните правой кнопкой мыши на ней и выберите пункт Additional Controls (Дополнительные элементы управления). В результате будет отображено диалоговое окно, показанное на рис. 13.15.

В диалоговом окне Additional Controls содержатся все элементы управления ActiveX, установленные в системе. Выберите элементы управления, которые необходимо добавить на панель инструментов. После этого щелкните на кнопке OK для добавления значков каждого из выбранных элементов управления.

Предупреждение

Не все элементы управления ActiveX, установленные в системе, поддерживаются пользовательскими диалоговыми окнами. Более того, большая их часть не поддерживается, к тому же некоторые элементы управления требуют лицензии на использование в приложениях. Если лицензия отсутствует, на экране появится сообщение об ошибке.

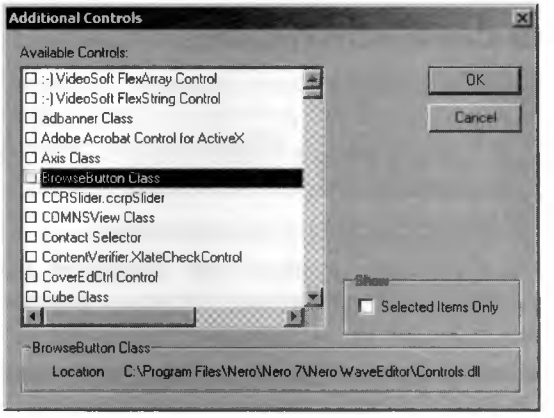


Рис. 13.15. В диалоговом окне Additional Controls можно найти дополнительные элементы управления ActiveX.

Создание шаблонов диалоговых окон

Зачастую при создании пользовательского диалогового окна каждый раз на форму добавляются одни и те же элементы управления. Например, все пользовательские диалоговые окна имеют два элемента управления `CommandButton`, используемых в качестве кнопок OK и Отмена. В предыдущем разделе рассматривались методы комбинирования элементов управления с целью получения одного элемента управления, обладающего функциями двух. Еще одной программной уловкой может служить шаблон диалогового окна, который при необходимости импортируется для последующего создания на его основе других проектов. Преимущество шаблонного подхода заключается в следующем: процедуры обработки событий сохраняются вместе с шаблоном.

Начните с создания пользовательского диалогового окна, содержащего все элементы управления и настройки, которые необходимо повторно использовать в других проектах. После этого убедитесь, что диалоговое окно выделено. Выберите команду File⇨Export File (Файл⇨Экспорт файла) (или нажмите комбинацию клавиш <Ctrl+E>). После этого на экране появится запрос на ввод имени файла. Затем для создания проекта на основе шаблона выполните команду File⇨Import File (Файл⇨Импорт файла), чтобы загрузить ранее сохраненное диалоговое окно.

Имитация диалоговых окон Excel

Внешний вид и поведение диалоговых окон Windows изменяются от программы к программе. При разработке приложений для Excel рекомендуется придерживаться стиля диалоговых окон Excel.

Наилучшим методом изучения эффективных способов создания диалоговых окон является повторное создание одного из стандартных диалоговых окон Excel. Например, удостоверьтесь, что вы правильно определили комбинации клавиш и активизировали элементы управления. Для создания копии одного диалогового окна Excel следует протестировать его в различных условиях. Один только анализ диалоговых окон Excel поможет улучшить познания в вопросах структуры окон и методов создания элементов управления.

Со временем вы убедитесь, что невозможно повторить отдельные диалоговые окна Excel даже с помощью VBA.

Вопросы для самоконтроля

Прежде чем предоставить самостоятельно созданное диалоговое окно на суд зрителей, необходимо удостовериться, что оно работает правильно. Ниже перечислены вопросы, ответы на которые позволяют выявить потенциальную проблему и ее причину.

- Все ли элементы управления одного типа имеют одинаковые размеры?
- Равномерно ли распределены элементы управления?
- Не возникают ли у зрителей трудности при просмотре диалогового окна? Если возникают, то следует перегруппировать элементы управления с помощью элемента управления `MultiPage`.
- Ко всем ли элементам управления можно получить доступ с помощью клавиатуры?
- Не повторяются ли комбинации клавиш?
- Правильно ли установлен порядок активизации элементов управления?
- Выполнит ли код VBA необходимые действия, если пользователь нажмет клавишу <Esc> или щелкнет на кнопке Отмена?
- Нет ли в тексте ошибок?
- Является ли правильным текст заголовка диалогового окна?
- Насколько правильно будет отображаться диалоговое окно при всех разрешениях экрана?
- Обеспечено ли логическое группирование элементов управления (в соответствии с выполняемой функцией)?
- Ограничивают ли элементы управления `ScrollBar` и `SpinButton` диапазон допустимых значений?
- Используются ли в форме `UserForm` элементы управления, которые могут быть не установлены в вашей системе?
- Правильно ли настроены элементы управления `ListBox` (`Single`, `Multi` и `Extended`)? За дополнительными сведениями об элементах управления `ListBox` обратитесь к главе 14.

Глава 14

Примеры пользовательских форм

В этой главе...

- Создание "меню" с помощью объекта `UserForm`
- Выбор диапазона в пользовательской форме
- Создание заставки
- Отключение кнопки закрытия пользовательского диалогового окна
- Изменение размера диалогового окна
- Масштабирование и прокрутка листа в пользовательском диалоговом окне
- Использование элемента управления `ListBox`
- Применение элемента управления `MultiPage`
- Использование внешних элементов управления
- Анимация элемента управления `Label`

В данной главе приведены полезные и информативные примеры использования пользовательских диалоговых окон.

Создание "меню" с помощью объекта `UserForm`

Иногда требуется применить пользовательское диалоговое окно в качестве меню. В данном разделе предлагаются два способа получения необходимого результата: с помощью элементов управления `CommandButton` и с помощью элемента управления `ListBox`.

Перекрестная ссылка

В главе 15 приведены примеры более сложных приемов работы с формами `UserForm`.

Использование элементов управления `CommandButton`

На рис. 14.1 показан пример объекта `UserForm`, в котором использован элемент управления `CommandButton` в качестве простого меню. Создание такого меню не вызывает особых трудностей, а код, обеспечивающий работу пользовательского диалогового окна, предельно прост. Каждый элемент управления `CommandButton` имеет собственную процедуру обработки событий. Например, представленная ниже процедура выполняется после щелчка на кнопке `CommandButton1`.

```
Private Sub CommandButton1_Click()
   Call Macro1
   Unload Me
End Sub
```

Эта процедура приводит к вызову макроса `Macro1` и закрытию диалогового окна `UserForm`. После щелчка на других кнопках (отличных от `CommandButton1`) вызываются похожие процедуры обработки событий.

Использование элемента управления `ListBox`

На рис. 14.2 показан пример создания меню с помощью элемента управления `ListBox`. Перед отображением пользовательского диалогового окна вызывается процедура обработки события `Initialize`. В приведенной ниже процедуре используется метод `AddItem` для добавления шести опций в элемент управления `ListBox`.

```
Private Sub UserForm_Initialize()
   With ListBox1
      .AddItem "Макрос1"
      .AddItem "Макрос2"
      .AddItem "Макрос3"
      .AddItem "Макрос4"
      .AddItem "Макрос5"
      .AddItem "Макрос6"
   End With
End Sub
```

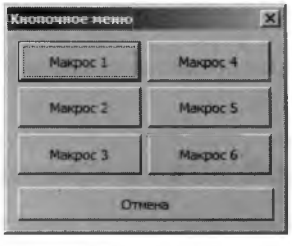


Рис. 14.1. В диалоговом окне отображается меню, в качестве пунктов которого используются элементы управления `CommandButton`

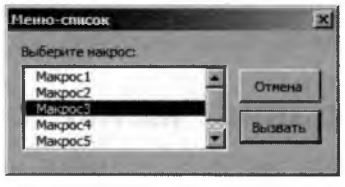


Рис. 14.2. В этом диалоговом окне отображается меню, составленное на основе элементов управления `ListBox`

Кроме того, процедура обработки события привязывается к кнопке Выполнить (Execute) для обработки щелчков на ней.

```
Private Sub ExecuteButton_Click()
  Select Case ListBox1.ListIndex
     Case -1
        MsgBox "Выберите макрос из списка."
        Exit Sub
     Case 0: Макрос1
     Case 1: Макрос2
     Case 2: Макрос3
     Case 3: Макрос4
     Case 4: Макрос5
     Case 5: Макрос6
  End Select
  UnIoad Me
End Sub
```

Данная процедура проверяет значение свойства `ListIndex` элемента управления `ListBox`, чтобы определить, какой элемент выбран в списке (если свойство `ListIndex` равно `-1`, то не выбран ни один из элементов). После этого запускается соответствующий макрос.

Компакт-диск

Два примера, рассмотренных в этом разделе, находятся на прилагаемом к книге компакт-диске в файле `userform menus.xlsm`.

Перекрестная ссылка

В главе 15 рассматривается похожий пример, в котором объект `UserForm` применяется для имитации панели инструментов.

Выбор диапазона в пользовательской форме

Некоторые встроенные диалоговые окна Excel предоставляют пользователю возможность выбирать диапазон. Например, диалоговое окно Подбор параметра (Goal Seek), для вызова которого достаточно выполнить команду Данные⇨Работа с данными⇨Анализ "что если"⇨Подбор параметра (Data⇨Data Tools⇨What-If Analysis⇨Goal Seek) запрашивает у пользователя два диапазона. Пользователь может или ввести имя диапазона непосредственно, или применить мышь для выделения диапазона на листе.

Пользовательское диалоговое окно также обеспечивает подобную функциональность. Это достигается с помощью элемента управления `RefEdit`. Данный элемент выглядит иначе, чем элемент выбора диапазона во встроенных диалоговых окнах Excel, однако работает точно так же. Если пользователь щелкнет на небольшой кнопке в правой части элемента управления, то диалоговое окно временно исчезнет, а на экране будет отображен небольшой указатель выбора диапазона (именно так все происходит при использовании встроенного диалогового окна Excel).

Примечание

К сожалению, элемент управления `RefEdit` не позволяет использовать специальные клавиши при выделении диапазона (например, невозможно выделить ячейки до конца столбца, нажав комбинацию клавиш <Shift+↓>). Кроме того, после щелчка мышью на маленькой кнопке в правой части элемента управления (для временного сокрытия диалогового окна) можно применять только выделения с помощью мыши. Клавиатуру в этом случае применять нельзя.

На рис. 14.3 представлено пользовательское диалоговое окно с добавленным элементом управления RefEdit. Это диалоговое окно выполняет простую математическую операцию над всеми не содержащими формул и непустыми ячейками указанного диапазона. Выполняемая операция задается активным переключателем OptionButton.

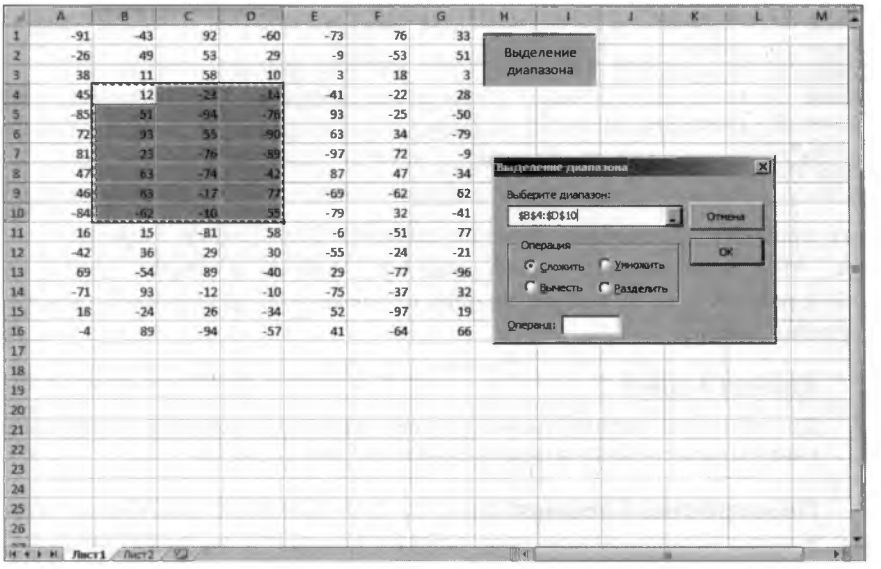


Рис. 14.3. С помощью элемента управления RefEdit можно выбрать диапазон

Компакт-диск

Рассматриваемый в этом разделе пример находится на прилагаемом к книге компакт-диске в файле range selection demo.xlsm.

Ниже изложены соображения, о которых следует помнить при использовании элемента управления RefEdit.

- Элемент управления RefEdit возвращает текстовую строку, которая представляет выбранный диапазон. Можно преобразовать эту строку в объект Range. Для этого используется следующий оператор:

  ```
  Set UserRange = Range(RefEdit1.Text)
  ```

- Удачной практикой считается инициализация элемента управления RefEdit для представления текущего выделения. Для этого в процедуре UserForm_Initialize воспользуйтесь таким оператором:

  ```
  RefEdit1.Text = ActiveWindow.RangeSelection.Address
  ```

- Для достижения наилучших результатов не помещайте элемент управления RefEdit внутри элемента Frame либо MultiPage. Это может привести к сбою в работе Excel.

- Элемент управления RefEdit не всегда возвращает действительный диапазон. Выделение диапазона указателем мыши — это один из способов присвоения значения данному элементу управления. Пользователь может ввести в поле любой текст, а также отредактировать или удалить уже отображаемый текст. Таким обра-

зом, предварительно необходимо убедиться, что диапазон является допустимым. Следующий код — это пример одного из способов проверки допустимости введенного значения. Если определено, что значение неправильное, то пользователю выдается сообщение, а элемент управления `RefEdit` становится активным, предоставляя возможность ввести корректный диапазон.

```
On Error Resume Next
Set UserRange = Range(RefEdit1.Text)
If Err.Number <> 0 Then
   MsgBox "Неправильный диапазон"
   RefEdit1.SetFocus
   Exit Sub
End If
On Error GoTo 0
```

- Пользователь может щелкнуть на вкладке одного из листов при выборе диапазона, применив элемент управления `RefEdit`. Поэтому не всегда выбранный диапазон находится на активном рабочем листе. Если пользователем выбран другой лист, то адрес диапазона указывается после имени листа, на котором этот диапазон находится. Пример приведен ниже.

  ```
  Лист2!$A$1:$C$4
  ```

- Если необходимо получить от пользователя выделение в виде одной ячейки, то можно указать верхнюю левую ячейку выделенного диапазона. В данном случае воспользуйтесь следующим оператором:

  ```
  Set OneCell = Range(RefEdit1.Text).Range("A1")
  ```

Перекрестная ссылка

Как отмечалось в главе 12, для выделения диапазона можно воспользоваться методом Excel `InputBox`.

Создание заставки

Некоторые разработчики предпочитают отображать определенную вступительную информацию при запуске приложения. Эта методика называется заставкой. Без сомнения, все пользователи видели заставку Excel, которая отображается несколько секунд при запуске программы.

В приложении Excel заставку можно создать с помощью пользовательского диалогового окна. В приведенном ниже примере реализуется автоматическое отображение заставки, которое исчезает по истечении пяти секунд.

Компакт-диск

На прилагаемом компакт-диске находится рабочая книга, демонстрирующая описанную в разделе процедуру. Соответствующий файл называется `splash screen.xlsm`.

Для создания заставки выполните следующие действия.

1. Создайте рабочую книгу.
2. Активизируйте редактор Visual Basic Editor (VBE) и вставьте пользовательское диалоговое окно в проект. Код в этом примере предполагает, что объект `UserForm` называется `UserForm1`.

3. Поместите любые необходимые элементы управления в только что созданное диалоговое окно `UserForm1`.

   Например, вам может понадобиться расположить элемент управления `Image`, который будет содержать логотип компании. На рис. 14.4 показан пример такого диалогового окна.

Рис. 14.4. Эта заставка на короткое время появляется на экране после открытия рабочей книги

4. Вставьте следующую процедуру в модуль кода для объекта `ЭтаКнига` (ThisWorkbook).

```
Private Sub Workbook_Open()
   UserForm1.Show
End Sub
```

5. Вставьте приведенную далее процедуру в модуль кода для объекта `UserForm1` (эта процедура обеспечивает пятисекундную задержку). Если нужно другое время задержки, измените значение аргумента функции `TimeValue`.

```
Private Sub UserForm_Activate()
   Application.OnTime Now + _
      TimeValue("00:00:05"), "KillTheForm"
End Sub
```

6. В общий модуль VBA вставьте следующую процедуру.

```
Private Sub KillTheForm()
   Unload UserForm1
End Sub
```

При открытии рабочей книги будет выполнена процедура `Workbook_Open` и появится диалоговое окно `UserForm` (п. 4). В этот момент генерируется событие `Activate`, которое приводит к выполнению процедуры `UserForm_Activate` (п. 5). Данная процедура использует метод `OnTime` объекта `Application` для выполнения процедуры `KillTheForm` в определенный момент

времени. Однако предварительно определена задержка в пять секунд с момента возникновения события Activate. Процедура KillTheForm просто выгружает диалоговое окно UserForm из памяти.

7. В качестве необязательного действия можно добавить элемент управления CommandButton с именем CancelButton, установить его свойство Cancel равным True и добавить представленную ниже процедуру обработки события в модуль кода формы UserForm.

```
Private Sub CancelButton_Click()
   KillTheForm
End Sub
```

Таким образом, пользователь сможет закрыть заставку, прежде чем пройдет указанное время задержки. Окно будет закрыто также в результате нажатия клавиши <Esc>. Эту небольшую кнопку можно разместить за другим объектом, чтобы она не была видна.

Предупреждение

Помните о том, что заставка не будет отображаться, если рабочая книга загружена не полностью. Другими словами, если нужно отобразить заставку только для того, чтобы пользователь не скучал во время загрузки рабочей книги, описанная выше методика не годится.

Совет

Для того чтобы выполнить VBA-процедуру при открытии документа, нужно так отобразить пользовательское диалоговое окно в немодальном режиме, чтобы код продолжал выполняться. Для этого измените процедуру Workbook_Open следующим образом.

```
Private Sub Workbook_Open()
   UserForm1.Show vbModeless
   '   другой код
End Sub
```

Отключение кнопки закрытия пользовательского диалогового окна

Если пользовательское диалоговое окно уже отображено на экране, щелчок на кнопке Закрыть (Close) (в правом верхнем углу) приведет к выгрузке формы UserForm из памяти. Иногда этого допускать нельзя. Например, иногда требуется, чтобы диалоговое окно UserForm закрывалось только после щелчка на специальной кнопке CommandButton. Несмотря на то что реально отключить кнопку Закрыть невозможно, вы вправе предотвратить закрытие диалогового окна, вызванное щелчком на этой кнопке. Для этого воспользуйтесь обработчиком события QueryClose.

Следующая процедура, которая расположена в модуле кода диалогового окна UserForm, выполняется перед закрытием формы (т.е. в момент возникновения события QueryClose).

```
Private Sub UserForm_QueryClose _
  (Cancel As Integer, CloseMode As Integer)
    If CloseMode = vbFormControlMenu Then
```

```
        MsgBox "Щелкните на кнопке OK для закрытия формы."
        Cancel = True
    End If
End Sub
```

Процедура `UserForm_QueryClose` использует два аргумента. Аргумент `CloseMode` содержит значение, которое указывает на причину возникновения события `QueryClose`. Если значение аргумента `CloseMode` равно `vbFormControlMenu` (встроенная константа), значит, пользователь щелкнул на кнопке **Закрыть**. В таком случае будет отображено сообщение; аргумент `Cancel` устанавливается равным `True`, и форма не закрывается.

Компакт-диск

Рассматриваемый в этом разделе пример можно найти на прилагаемом к книге компакт-диске в файле `queryclose demo.xlsm`.

Примечание

Имейте в виду, что пользователь может нажать клавиши <Ctrl+Break>, прекратив тем самым выполнение макроса. В рассматриваемом примере нажатие клавиш <Ctrl+Break> во время отображения формы `UserForm` на экране приведет к тому, что пользовательское диалоговое окно будет закрыто. Во избежание этого выполните следующий оператор до отображения пользовательского диалогового окна:

```
Application.EnableCancelKey = xlDisabled
```

Прежде чем добавить этот оператор, удостоверьтесь, что в приложении нет ошибок. В противном случае возникает опасность формирования бесконечного цикла.

Изменение размера диалогового окна

Во многих приложениях используются окна, которые могут изменять собственные размеры. Например, высота диалогового окна Excel **Найти и заменить** (Find and Replace), которое отображается после выбора команды **Главная⇨Редактирование⇨Найти и выделить⇨Заменить** (Home⇨Editing⇨Find & Select⇨Replace), увеличивается после щелчка на кнопке **Параметры** (Options).

В рассматриваемом в этом разделе примере демонстрируется предоставление пользовательскому диалоговому окну возможности динамически изменять свой размер. Изменение размера диалогового окна осуществляется путем изменения значений свойств `Width` и `Height` объекта `UserForm`.

Перекрестная ссылка

Обратитесь к главе 15, в которой рассматривается пример формы `UserForm`, размер которой изменяется путем перетаскивания нижнего правого угла окна.

На рис. 14.5 показано первоначальное диалоговое окно, а на рис. 14.6 показано это же окно после щелчка на кнопке **Параметры** (Options). Обратите внимание на то, что надпись на кнопке изменяется в зависимости от размера диалогового окна.

Рис. 14.5. Диалоговое окно, отображаемое в стандартном режиме

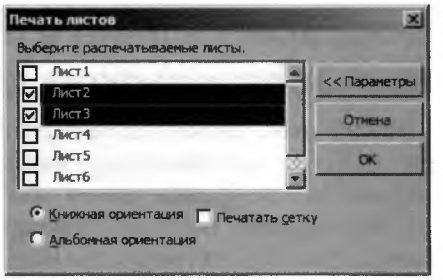


Рис. 14.6. Увеличенное диалоговое окно, отображающее дополнительные параметры

Создавая пользовательское диалоговое окно, определите его максимальный размер, чтобы получить доступ ко всем элементам управления. После этого воспользуйтесь процедурой `UserForm_Initialize` для установки размеров диалогового окна по умолчанию.

В коде применяются две константы, определенные в верхней части модуля.

```
Const SmallSize As Integer = 124
Const LargeSize As Integer = 164
```

Описанный пример предназначен для печати рабочих листов активной книги. Он позволяет пользователю указать листы, которые необходимо распечатать. Ниже приведена процедура обработки события, которая выполняется после щелчка на кнопке Параметры.

```
Private Sub OptionsButton_Click()
   If OptionsButton.Caption = "Параметры >>" Then
      Me.Height = LargeSize
      OptionsButton.Caption = "<< Параметры"
   Else
      Me.Height = SmallSize
      OptionsButton.Caption = "Параметры >>"
   End If
End Sub
```

Эта процедура проверяет значение свойства `Caption` объекта `CommandButton` и устанавливает значение свойства `Height` объекта `UserForm` в соответствии с полученным значением свойства `Caption`.

Примечание

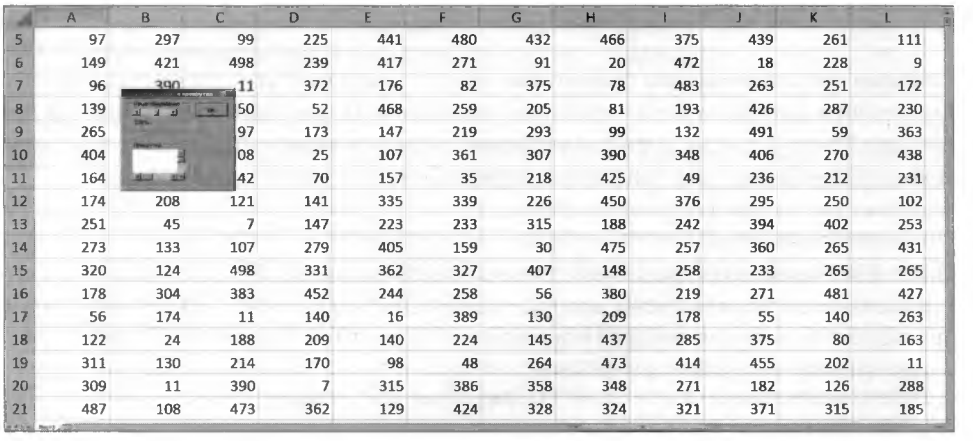
Если элементы управления не отображаются из-за того, что находятся за пределами границы диалогового окна, соответствующие этим элементам управления комбинации клавиш будут продолжать функционировать. В рассматриваемом примере пользователь может нажать клавиши <Alt+L> (для выбора альбомной ориентации страницы), даже если соответствующий элемент управления не отображается на экране. Для блокирования доступа к невидимым элементам управления напишите код, который отключает элементы управления, если они скрыты.

Компакт-диск

Рассматриваемый в этом разделе пример можно найти на прилагаемом к книге компакт-диске в файле `change userform size.xlsm`.

Масштабирование и прокрутка листа в пользовательском диалоговом окне

Пример, приведенный в этом разделе, демонстрирует применение элемента управления `ScrollBar` для прокрутки и масштабирования листа при активном диалоговом окне. На рис. 14.7 отображен пример окна изменения масштаба рабочего листа (в диапазоне от 100 до 400%) с помощью элемента управления `ScrollBar`, находящегося в верхней части диалогового окна. Два элемента управления `ScrollBar` в нижней части диалогового окна позволяют прокручивать лист по горизонтали и по вертикали.



	A	B	C	D	E	F	G	H	I	J	K	L
5	97	297	99	225	441	480	432	466	375	439	261	111
6	149	421	498	239	417	271	91	20	472	18	228	9
7	96	390	11	372	176	82	375	78	483	263	251	172
8	139		50	52	468	259	205	81	193	426	287	230
9	265		97	173	147	219	293	99	132	491	59	363
10	404		08	25	107	361	307	390	348	406	270	438
11	164		42	70	157	35	218	425	49	236	212	231
12	174	208	121	141	335	339	226	450	376	295	250	102
13	251	45	7	147	223	233	315	188	242	394	402	253
14	273	133	107	279	405	159	30	475	257	360	265	431
15	320	124	498	331	362	327	407	148	258	233	265	265
16	178	304	383	452	244	258	56	380	219	271	481	427
17	56	174	11	140	16	389	130	209	178	55	140	263
18	122	24	188	209	140	224	145	437	285	375	80	163
19	311	130	214	170	98	48	264	473	414	455	202	11
20	309	11	390	7	315	386	358	348	271	182	126	288
21	487	108	473	362	129	424	328	324	321	371	315	185

Рис. 14.7. Элементы управления `ScrollBar` позволяют прокручивать лист и изменять его масштаб

Компакт-диск

Рассматриваемый в этом разделе пример находится на прилагаемом к книге компакт-диске в файле `zoom and scroll sheet.xlsm`.

Просмотрев код этого примера, вы отметите для себя его простоту. Элементы управления инициализируются в процедуре `UserForm_Initialize`, как показано ниже.

```
Private Sub UserForm_Initialize()
    LabelZoom.Caption = ActiveWindow.Zoom & "%"
'   Масштабирование
    With ScrollBarZoom
        .Min = 10
        .Max = 400
        .SmallChange = 1
        .LargeChange = 10
        .Value = ActiveWindow.Zoom
    End With

'   Горизонтальная прокрутка
    With ScrollBarColumns
        .Min = 1
        .Max = ActiveSheet.UsedRange.Columns.Count
        .Value = ActiveWindow.ScrollColumn
        .LargeChange = 25
        .SmallChange = 1
    End With
```

```
' Вертикальная прокрутка
  With ScrollBarRows
     .Min = 1
     .Max = ActiveSheet.UsedRange.Rows.Count
     .Value = ActiveWindow.ScrollRow
     .LargeChange = 25
     .SmallChange = 1
  End With
End Sub
```

Эта процедура позволяет устанавливать значения различных свойств элементов управления `ScrollBar`. Значения определяются на основе данных, полученных из активного окна. При использовании элемента управления `ScrollBarZoom` выполняется процедура `ScrollBarZoom_Change` (которая приведена ниже). Она устанавливает значение свойства `Zoom` объекта `ActiveWindow` равным значению свойства `Value` элемента управления `ScrollBar`. Кроме того, изменяется текст подписи, которая представляет текущий масштаб рабочего листа.

```
Private Sub ScrollBarZoom_Change()
  With ActiveWindow
     .Zoom = ScrollBarZoom.Value
     LabelZoom = .Zoom & "%"
     End With
End Sub
```

Прокрутка листа осуществляется с помощью двух процедур. Эти процедуры устанавливают значение свойств `ScrollRow` и `ScrollColumns` объекта `ActiveWindow` равными значениям свойств `Value` элементов управления `ScrollBar`.

```
Private Sub ScrollBarColumns_Change()
  ActiveWindow.ScrollColumn = ScrollBarColumns.Value
End Sub

Private Sub ScrollBarRows_Change()
  ActiveWindow.ScrollRow = ScrollBarRows.Value
End Sub
```

Совет

Вместо события `Change` в предыдущих процедурах лучше обратиться к событию `Scroll`. Разница заключается в том, что последнее событие наступает при прокрутке и масштабировании документа. Для использования события `Scroll` просто включите `Change` в название процедуры `Scroll`.

Использование элемента управления `ListBox`

Элемент управления `ListBox` довольно гибкий в использовании, но работа с ним может оказаться достаточно сложной. В данном разделе приведен ряд простых примеров распространенных методик использования элемента управления `ListBox`.

Примечание

В большинстве случаев методы, описанные в этом разделе, могут применяться и для работы с элементом управления `ComboBox`.

Ниже изложены рекомендации, о которых необходимо помнить при работе с элементом управления ListBox. Примеры данного раздела наглядно иллюстрируют эти советы.

- Опции списка элемента управления ListBox могут извлекаться из диапазона ячеек (определяемого свойством RowSource) или добавляться с помощью VBA (для этого используется метод AddItem).
- Элемент управления ListBox может быть применен для выделения одной или нескольких опций. Соответствующее поведение определяется значением свойства MultiSelect.
- Если элемент управления ListBox не настроен на выделение нескольких опций, то значение элемента управления ListBox может связываться с ячейкой листа с помощью свойства ControlSource.
- Элемент управления ListBox может отображаться без предварительно выбранной опции (для этого необходимо установить свойство ListIndex равным -1). Но как только пользователь выделит одну из опций списка, отменить выделение будет невозможно. Исключение из этого правила — значение свойства MultiSelect равно True.
- Элемент управления ListBox может содержать несколько столбцов (что указывается в свойстве ColumnCount) и даже описательные заголовки (для этого используется свойство ColumnHeads).
- Вертикальный размер элемента управления ListBox, помещенного в пользовательское диалоговое окно, не всегда совпадает с вертикальным размером объекта UserForm на экране.
- Опции списка элемента управления ListBox могут отображаться в виде флажков, если разрешено выделение нескольких элементов, или в виде переключателей, если поддерживается только единичное выделение. Это поведение определяется значением свойства ListStyle.

Для получения полной информации о свойствах и методах элемента управления ListBox обратитесь к интерактивной справке.

Добавление опций в элемент управления ListBox

Перед отображением пользовательского диалогового окна, которое содержит элемент управления ListBox, следует заполнить элемент управления ListBox необходимыми опциями. Элемент управления ListBox заполняется на этапе разработки проекта посредством указания диапазона ячеек, которые содержат необходимые данные. Его также можно заполнить на этапе выполнения, воспользовавшись кодом VBA для добавления всех опций списка.

В двух примерах из настоящего раздела предполагается следующее:

- используется диалоговое окно UserForm с именем UserForm1;
- диалоговое окно UserForm1 содержит элемент управления ListBox, который называется ListBox1;
- рабочая книга содержит лист Лист1, в диапазоне A1:A12 которого определены опции, отображаемые в элементе управления ListBox.

Добавление опций в элемент управления ListBox на этапе разработки

Для добавления опций в элемент управления ListBox на этапе разработки необходимо, чтобы они хранились в диапазоне ячеек рабочей книги. Воспользуйтесь свойством RowSource для указания диапазона, который содержит опции элемента управления ListBox. На рис. 14.8 показано окно Properties (Свойства) для элемента управления ListBox. Свойство RowSource установлено равным Лист1!A1:A12. Когда диалоговое окно UserForm отображается на экране, элемент управления ListBox содержит двенадцать опций из этого диапазона. Опции добавляются в элемент управления ListBox на этапе разработки, сразу после того, как диапазон определяется в качестве значения свойства RowSource.

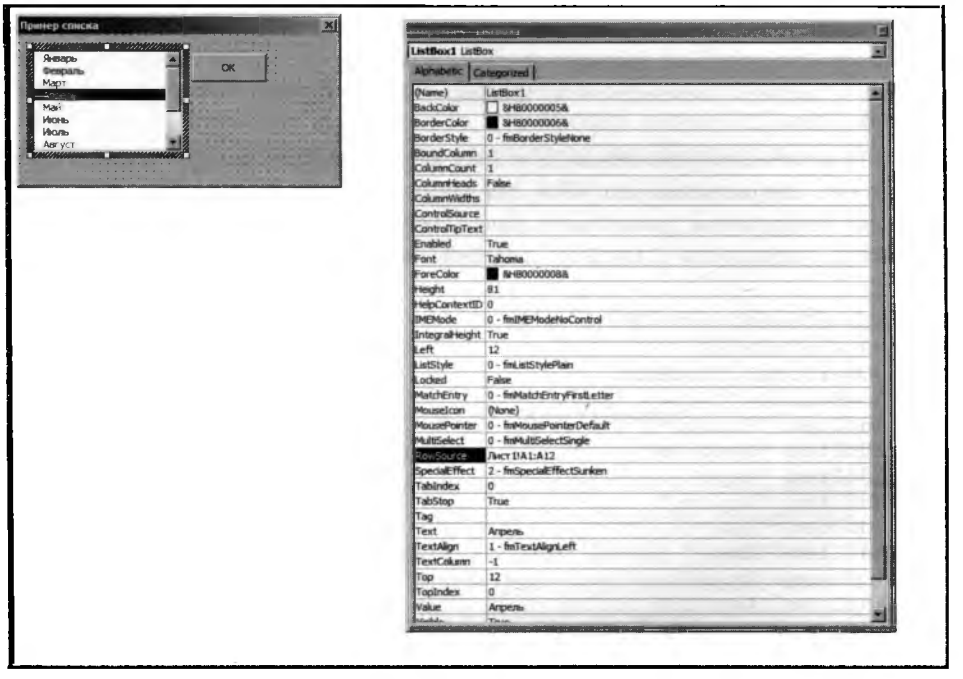


Рис. 14.8. Установка свойства RowSource на этапе разработки

Предупреждение

Удостоверьтесь, что в значении свойства RowSource присутствует имя листа. В противном случае элемент управления ListBox будет применять указанный диапазон в активном рабочем листе. Иногда следует идентифицировать диапазон максимально точно, указав даже имя рабочей книги, например:

```
[budget.xlsx]Лист1!A1:A12
```

Лучше всего сначала определить имя для диапазона, а затем включить его в состав разработанного кода. Это гарантирует использование требуемого диапазона, даже если были добавлены или удалены строки, не входящие в диапазон.

Добавление опций в элемент управления ListBox на этапе выполнения

Чтобы добавить опции элемента управления ListBox на этапе выполнения, необходимо реализовать следующее:

- с помощью кода определить значение свойства RowSource, чтобы указать диапазон, хранящий необходимые данные;
- создать код, использующий метод AddItem для добавления опций в элемент управления ListBox.

Как и следовало ожидать, значение свойства RowSource устанавливается с помощью кода, а не в окне Properties. Например, представленная далее процедура устанавливает значения свойства RowSource элемента управления ListBox перед тем, как отображается диалоговое окно UserForm. В этом случае опции состоят из значений в ячейках диапазона Categories рабочего листа Budget.

```
UserForm1.ListBox1.RowSource = "Budget!Categories"
UserForm1.Show
```

Если опции элемента управления ListBox не содержатся в диапазоне ячеек листа, то можно создать специальный код VBA для заполнения элемента управления ListBox перед кодом отображения диалогового окна. Следующая процедура заполняет окно списка элемента управления ListBox названиями месяцев года с помощью метода AddItem.

```
Sub ShowUserForm2()
'  Заполнение списка
   With UserForm1.ListBoxI
      .RowSource=""
      .AddItem "Январь"
      .AddItem "Февраль"
      .AddItem "Март"
      .AddItem "Апрель"
      .AddItem "Май"
      .AddItem "Июнь"
      .AddItem "Июль"
      .AddItem "Август"
      .AddItem "Сентябрь"
      .AddItem "Октябрь"
      .AddItem "Ноябрь"
      .AddItem "Декабрь"
   End With
   UserForm1.Show
End Sub
```

Предупреждение

В предыдущем коде свойство RowSource сначала приравнивалось к пустой строке. Таким образом предотвращалась потенциальная ошибка, которая появляется в том случае, когда в окне Properties свойство RowSource элемента управления ListBox имеет определенное значение. Если попытаться добавить опции в элемент управления ListBox с ненулевым значением свойства RowSource, появится сообщение об ошибке "permission denied" (отсутствуют права доступа).

Можно также использовать метод AddItem для выборки опций элемента управления ListBox, хранящихся в диапазоне ячеек. Ниже рассмотрен пример заполнения элемента управления ListBox содержимым диапазона A1:A12 листа Лист1.

```
For Row = 1 To 12
   UserForm1.ListBox1.AddItem Sheets("Лист1").Cells(Row, 1)
Next Row
```

Использование свойства List еще больше облегчает решение задачи. Приведенный ниже оператор дает тот же эффект, что и предыдущий цикл For Next.

```
UserForm1.ListBox1.List = Application.Transpose(Sheets( _
   "Лист1").Range("A1:A12"))
```

Обратите внимание, что использовалась функция Transpose, поскольку свойство List настроено на ввод массива-строки, в то время как диапазон ячеек представляет собой массив-столбец.

Если данные хранятся в одномерном (векторном) массиве, можно воспользоваться свойством List. Например, предположим, что имеется массив MyList, который содержит 50 элементов. Приведенный ниже оператор создает 50-элементный список в элементе управления ListBox1.

```
UserForm1.ListBox1.List = MyList
```

Компакт-диск

Примеры из этого раздела находятся на прилагаемом к книге компакт-диске в файле fill listbox.xlsm.

Добавление в элемент управления ListBox только уникальных элементов

В определенных случаях возникает необходимость в заполнении элемента управления ListBox уникальными (неповторяющимися) опциями из существующего списка. Предположим, у нас есть лист, который содержит данные о заказчиках. В одном из столбцов могут содержаться названия штатов (рис. 14.9). Необходимо заполнить элемент управления ListBox названиями штатов, в которых проживают потребители, исключив при этом дублирование названий штатов.

Один из методов заполнения предполагает использование объекта Collection. Элементы в объект Collection добавляются с помощью следующего синтаксиса:

```
object.Add item, key, before, after
```

Аргумент key (если он используется) содержит уникальную текстовую строку, которая необходима для получения доступа к элементам коллекции. Если к коллекции добавить неуникальный ключ, то возникнет ошибка, и в результате элемент добавлен не будет. Этим можно воспользоваться и создать коллекцию, которая содержит только уникальные элементы.

Представленная далее процедура демонстрирует использование этого способа. Процедура начинается с объявления нового объекта коллекции — NoDupes. Предполагается, что диапазон, называющийся Data, содержит список элементов, часть которых повторяется.

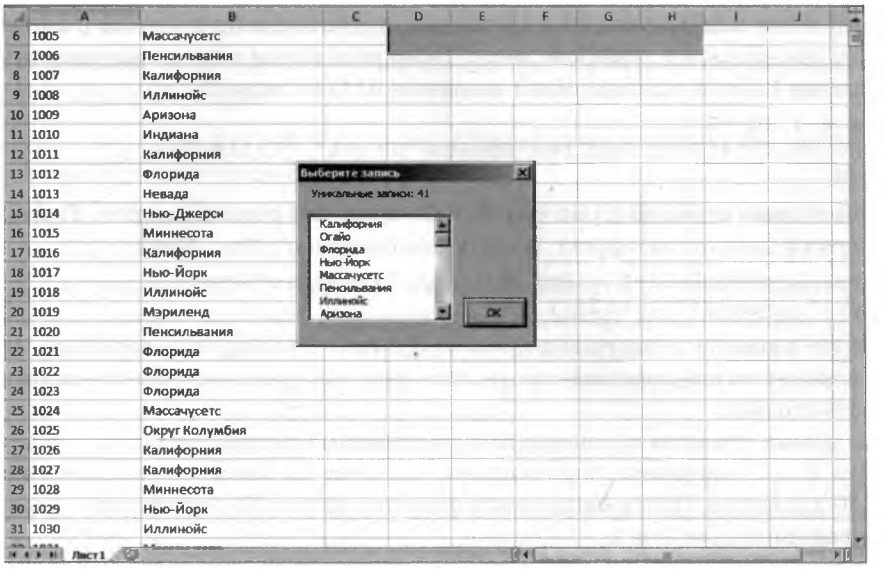


Рис. 14.9. Объект `Collection` применяется для заполнения элемента управления `ListBox` уникальными значениями из столбца B

В коде циклически просматриваются ячейки диапазона, и в коллекцию `NoDupes` добавляются значения только уникальных ячеек. Кроме того, значение ячейки (преобразованное в строку) используется в качестве значения аргумента `key`. Применение оператора `On Error Resume Next` приводит к тому, что в коде VBA игнорируется ошибка, которая возникает при добавлении в коллекцию неуникального ключа. Если возникает ошибка, элемент в коллекцию не добавляется — это именно то поведение, которого необходимо добиться. Затем процедура передает элементы коллекции `NoDupes` в элемент управления `ListBox`. В диалоговом окне `UserForm` также содержится подпись, которая указывает количество уникальных элементов коллекции.

```
Sub RemoveDuplicates1()
   Dim AllCells As Range, Cell As Range
   Dim NoDupes As New Collection

   On Error Resume Next
   For Each Cell In Range("State")
      NoDupes.Add Cell.Value, CStr(Cell.Value)
   Next Cell
   On Error GoTo 0
'  Добавление уникальных элементов в элемент управления ListBox
   For Each Item In NoDupes
      UserForm1.ListBox1.AddItem Item
   Next Item
'  Отображение количества элементов
   UserForm1.Label1.Caption = "Уникальные элементы: " & NoDupes.Count
'  Отображение диалогового окна UserForm
   UserForm1.Show
End Sub
```

Компакт-диск

Рассматриваемый в этом разделе пример находится на прилагаемом к книге компакт-диске (в файле `listbox unique items1.xlsm`). В рабочей книге `listbox unique items2.xlsm` находится более сложный пример, выводящий на экран отсортированные элементы.

Определение выделенного элемента списка

В примерах, рассмотренных в предыдущих разделах, отображалось диалоговое окно `UserForm` с элементом управления `ListBox`, который содержит список из нескольких элементов. Эти процедуры не включают главной функции: определения опции или опций, которые выбраны пользователем.

Примечание

В дальнейшем будет рассматриваться элемент управления `ListBox` с одним выделенным элементом списка — его свойство `MultiSelect` должно иметь значение `0`.

Чтобы определить, какой элемент списка выбран, необходимо узнать значение свойства `Value` элемента управления `ListBox`. Оператор, показанный ниже, отображает текст выделенного в объекте `ListBox1` элемента.

```
MsgBox ListBox1.Value
```

Если не выбран ни один элемент списка, то выполнение оператора приведет к возникновению ошибки.

Чтобы узнать расположение выделенного элемента в списке (а не только его содержимое), воспользуйтесь значением свойства `ListIndex` элемента управления `ListBox`.

В следующем примере демонстрируется простое окно сообщения, в котором указан номер выделенной позиции элемента управления `ListBox`.

```
MsgBox "Вы выбрали позицию #" & ListBoxI.ListIndex
```

Если не выделен ни один элемент списка, свойство `ListIndex` возвращает значение `-1`.

Примечание

Нумерация позиций в элементе управления `ListBox` начинается с `0`, а не с `1`. Таким образом, значение свойства `ListIndex` для первого элемента будет `0`, а для последнего элемента значение свойства соответствует значению свойства `ListCount`, равному `1`.

Определение нескольких выделенных элементов списка

Свойство `MultiSelect` элемента управления `ListBox` может принимать одно из трех приведенных ниже значений.

- `0` (`fmMultiSelectSingle`). Может быть выбран только один элемент. Эта установка задана по умолчанию.
- `1` (`fmMultiSelectMulti`). Нажмите клавишу пробела или щелкните мышью для выделения (отмены выделения) элементов в списке.

- 2 (fmMultiSelectExtended). Удерживая клавишу <Shift>, щелкайте мышью для расширения области выделения от предыдущего до текущего элемента. Для расширения выделенной области можно также удерживать нажатой клавишу <Shift> и нажимать одну из клавиш управления курсором.

Если элемент управления ListBox разрешает выделение нескольких элементов (свойство MultiSelect равно 1 или 2), попытка доступа к свойству ListIndex либо Value приведет к ошибке. В этом случае лучше использовать свойство Selected, возвращающее массив, первый элемент которого имеет индекс 0. Например, следующий оператор возвращает значение True, если выделен первый элемент в списке ListBox.

```
MsgBox ListBox1.Selected(0)
```

Компакт-диск

На прилагаемом компакт-диске находится рабочая книга, которая демонстрирует способы идентификации выбранных опций элемента управления ListBox. Эта рабочая книга включает элементы управления ListBox, которые допускают однократное и многократное выделение опций, и находится в файле listbox selected items.xlsm.

Следующий код, "заимствованный" из рабочей книги, находящейся на прилагаемом компакт-диске, демонстрирует циклический обход каждой опции элемента управления ListBox. Если опция выделена, соответствующий ей текст добавляется в переменную Msg. И наконец, названия всех выделенных опций отображаются в окне сообщения.

```
Private Sub OKButton_Click()
   Msg = ""
   For i = 0 To ListBox1.ListCount - 1
      If ListBox1.Selected(i) Then _
         Msg = Msg & ListBox1.List(i) & vbCrLf
   Next i
   MsgBox "Вы выбрали: " & vbCrLf & Msg
   Unload Me
End Sub
```

На рис. 14.10 показан результат выделения нескольких опций элемента управления ListBox.

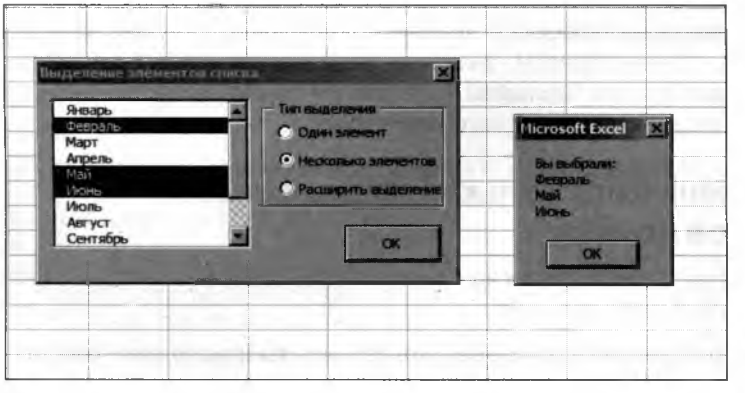


Рис. 14.10. В окне сообщения отображается список выделенных опций элемента управления ListBox

Несколько списков в одном элементе управления ListBox

В приведенном ниже примере демонстрируется создание элемента управления ListBox, изменяющего свое содержимое в зависимости от того, какие переключатели OptionButton установил пользователь.

На рис. 14.11 показан пример диалогового окна UserForm. Элемент управления ListBox получает список значений из диапазона на листе. Процедуры, обрабатывающие событие Click для элементов управления OptionButton, устанавливают значение свойства RowSource элемента управления ListBox равным необходимому диапазону. Одна из таких процедур представлена далее.

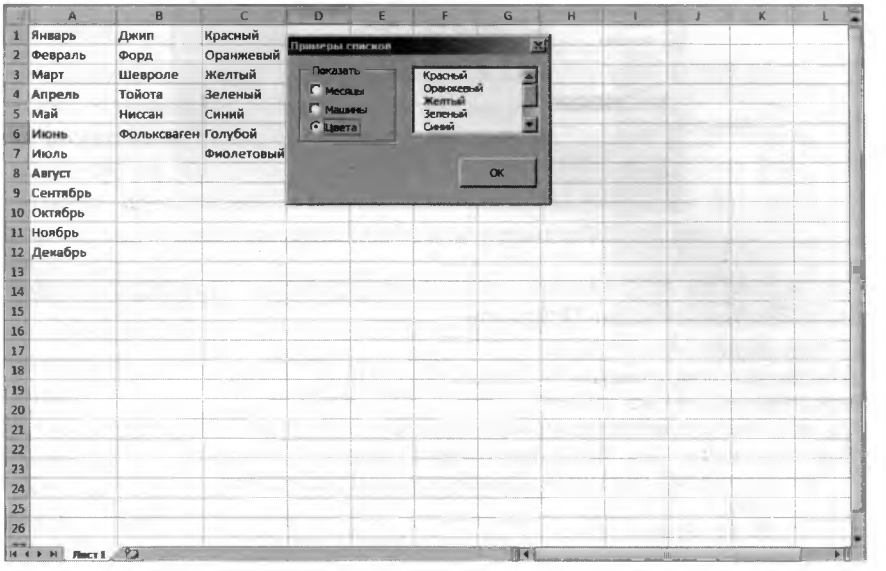


Рис. 14.11. Содержимое элемента управления ListBox зависит от того, какой элемент управления OptionButton выбран в настоящий момент

```
Private Sub obMonths_Click()
   ListBox1.RowSource = "Лист1!Месяцы"
End Sub
```

Щелчок на элементе управления OptionButton, называющемся obMonths, приводит к изменению значения свойства RowSource элемента управления ListBox, что заставляет его использовать диапазон Месяцы на листе Лист1.

Компакт-диск

Рассматриваемый в этом разделе пример можно найти на прилагаемом к книге компакт-диске в файле listbox multiple lists.xlsm.

Передача опций элемента управления ListBox

В некоторых приложениях требуется выбрать несколько элементов списка. Зачастую следует создать список на основе выделенных элементов. Примером такой ситуации мо-

жет быть вкладка **Панель быстрого доступа** (Quick Access Toolbar) в диалоговом окне **Параметры Excel** (Excel Options).

На рис. 14.12 показано диалоговое окно с двумя элементами управления `ListBox`. Кнопка **Добавить** добавляет элемент, выделенный в левом элементе управления `ListBox`, в правый элемент управления `ListBox`. Кнопка **Удалить** удаляет выделенный элемент из правого списка. Флажок определяет поведение при добавлении в список повторяющихся элементов. Если флажок **Разрешить дублирование** не установлен, то в случае, если пользователь попытается добавить элемент, который уже присутствует в списке, будет отображено окно сообщения.

Рис. 14.12. Построение одного списка на основе другого

Код этого примера на удивление прост. Ниже приведена процедура, которая выполняется после щелчка на кнопке **Добавить**.

```
Private Sub AddButton_Click()
   If ListBox1.ListIndex = -1 Then Exit Sub
   If Not cbDuplicates Then
'     Проверить существование элемента
      For i = 0 To ListBox2.ListCount - 1
         If ListBox1.Value = ListBox2.List(i) Then
            Beep
            Exit Sub
         End If
      Next i
   End If
   ListBox2.AddItem ListBox1.Value
End Sub
```

Код для управления кнопкой **Удалить** еще проще.

```
Private Sub RemoveButton_Click()
   If ListBox2.ListIndex = -1 Then Exit Sub
   ListBox2.RemoveItem ListBox2.ListIndex
End Sub
```

Обратите внимание, что обе процедуры проверяют существование выделенного элемента. Если значение свойства `ListIndex` элемента управления `ListBox` равно `-1`, значит, не выделен ни один элемент. В результате процедура завершается.

Этот пример также включает две дополнительные процедуры, которые проверяют состояние кнопки **Удалить** (активна или неактивна). При этом вызываются соответствующие события в результате ввода данных в элемент управления `ListBox` (с помощью клавиатуры или щелчка мышью). В результате кнопка **Удалить** активизируется только в том случае, когда пользователь работает с элементом управления `ListBox2`.

```
Private Sub ListBox1_Enter()
   RemoveButton.Enabled = False
End Sub

Private Sub ListBox2_Enter()
   RemoveButton.Enabled = True
End Sub
```

Компакт-диск

Рассматриваемый в этом разделе пример находится на прилагаемом к книге компакт-диске в файле `listbox item transfer.xlsm`.

Перемещение опций в списке элементов управления `ListBox`

Часто порядок, в котором расположены опции в списке, имеет значение. Из данного раздела вы узнаете, как можно перемещать опции вверх или вниз в списке элемента управления `ListBox`. В VBE подобная техника применяется для контроля порядка активизации опций в окне `UserForm` (щелкните правой кнопкой мыши в окне `UserForm` и в контекстном меню выберите команду Tab Order).

На рис. 14.13 показано диалоговое окно с элементом управления `ListBox` и двумя элементами управления `CommandButton`. Щелчок на кнопке Вверх приведет к перемещению выделенной опции вверх по списку элемента управления `ListBox`. Щелчок на кнопке Вниз приведет к перемещению выделенной опции вниз по списку.

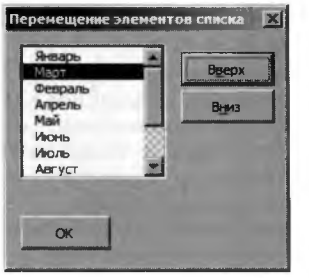


Рис. 14.13. Перемещаться вверх или вниз по списку можно с помощью специально предназначенных для этого кнопок

Компакт-диск

Этот пример находится на прилагаемом к книге компакт-диске в файле `listbox move items.xlsm`.

Ниже приведены процедуры-обработчики событий для двух элементов управления `CommandButton`.

```
Private Sub MoveUpButton_Click()
   Dim NumItems As Integer, i As Integer, ItemNum As Integer
   Dim TempItem As String, TempList()
   If ListBox1.ListIndex <= 0 Then Exit Sub
   NumItems = ListBox1.ListCount
   Dim TempList()
   ReDim TempList(0 To NumItems - 1)
'  Заполнение массива элементами списка
   For i = 0 To NumItems - 1
      TempList(i) = ListBox1.List(i)
   Next i
'  Выбранный элемент
   ItemNum = ListBox1.ListIndex
'  Обмен элементами
```

```
    TempItem = TempList(ItemNum)
    TempList(ItemNum) = TempList(ItemNum - 1)
    TempList(ItemNum - 1) = TempItem
    ListBox1.List = TempList
'   Изменение индекса списка
    ListBox1.ListIndex = ItemNum - 1
End Sub

Private Sub MoveDownButton_Click()
    Dim NumItems As Integer, i As Integer, ItemNum As Integer
    Dim TempItem As String, TempList()
    If ListBox1.ListIndex = ListBox1.ListCount - 1 Then Exit Sub
    NumItems = ListBox1.ListCount
    Dim TempList()
    ReDim TempList(0 To NumItems - 1)
'   Заполнение массива элементами списка
    For i = 0 To NumItems - 1
       TempList(i) = ListBox1.List(i)
    Next i
'   Выделенный элемент
    ItemNum = ListBox1.ListIndex
'   Обмен элементами
    TempItem = TempList(ItemNum)
    TempList(ItemNum) = TempList(ItemNum + 1)
    TempList(ItemNum + 1) = TempItem
    ListBox1.List = TempList
'   Изменение индекса списка
    ListBox1.ListIndex = ItemNum + 1
End Sub
```

Работа с многоколоночными элементами управления ListBox

Как правило, элемент управления ListBox содержит один столбец, в котором отображается один список. Однако можно создать элемент управления ListBox, который содержит несколько столбцов, а иногда — даже несколько столбцов с заголовками. На рис. 14.14 отображен элемент управления ListBox с несколькими столбцами, который получает данные из диапазона ячеек рабочего листа.

Компакт-диск

Рассмотренный в этом разделе пример находится на прилагаемом к книге компакт-диске в файле listbox multicolumn1.xlsm.

Для того чтобы создать элемент управления ListBox с несколькими столбцами, в которые заносятся данные, хранимые в диапазоне ячеек листа, выполните следующие действия.

1. Удостоверьтесь, что свойству ColumnCount элемента управления ListBox присвоено правильное значение, которое соответствует количеству столбцов в элементе управления.
2. Укажите правильный исходный диапазон данных из нескольких столбцов, присвоив соответствующее значение свойству RowSource элемента управления ListBox.

3. Если необходимо отобразить заголовки столбцов, присвойте свойству `ColumnHeads` значение `True`. Не включайте заголовки столбцов в диапазон рабочего листа, указанный в свойстве `RowSource`. VBA автоматически использует для них строку, которая находится сразу над строкой, указанной в значении свойства `RowSource`.

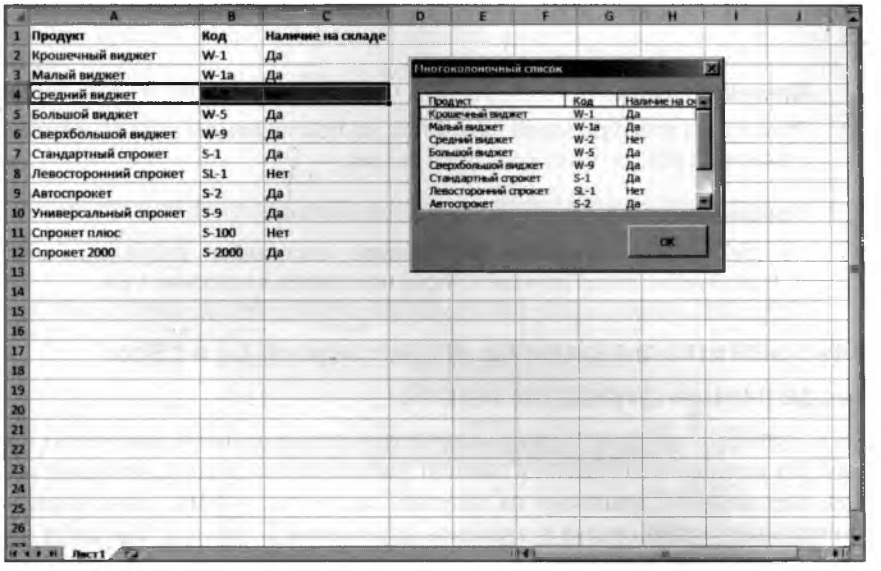


Рис. 14.14. Этот элемент управления `ListBox` отображает трехколоночный список с заголовками колонок

4. Измените ширину столбцов, присвоив свойству `ColumnWidths` значения, которые указываются в пунктах (1/72 часть дюйма) и разделены точками с запятой. Например, следующее значение свойства `ColumnWidths` определяет ширину трех столбцов списка элемента управления `ListBox` — `100;40;30`.
5. Укажите столбец в качестве значения свойства `BoundColumn`. Это свойство определяет столбец, на который указывает ссылка при обращении к свойству `Value` элемента управления `ListBox`.

Чтобы заполнить элемент управления `ListBox` данными из нескольких столбцов без использования диапазона, необходимо создать двумерный массив, а затем присвоить массив свойству `List` элемента управления `ListBox`. Следующие операторы демонстрируют применение двумерного массива (размером 12×2) под названием `Data`. Двухколоночный список `ListBox` отображает названия месяцев в столбце 1, а количество дней — в столбце 2 (рис. 14.15). Обратите внимание, что процедура присваивает свойству `ColumnCount` значение 2.

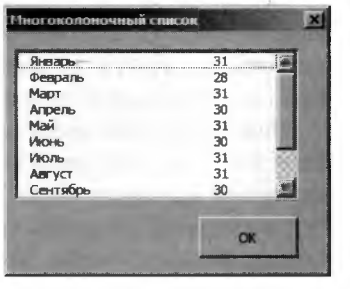


Рис. 14.15. Двухколоночный список `ListBox`, заполненный данными двумерного массива

```
Private Sub UserForm_Initialize()
' Заполнить элемент управления ListBox
  Dim Data(1 To 12, 1 To 2)
```

```
    For i = 1 To 12
        Data(i, 1) = Format(DateSerial(2007, i, 1), "mmmm")
    Next i
    For i = 1 To 12
        Data(i, 2) = Day(DateSerial(2007, i + 1, 1) - 1)
    Next i
    ListBox1.ColumnCount = 2
    ListBox1.List = Data
End Sub
```

Компакт-диск

Пример из этого раздела находится на прилагаемом к книге компакт-диске в файле `listbox multicolumn2.xlsm`.

Примечание

Не существует способа определить заголовки столбцов в свойстве `ColumnHeads`, когда данные списка находятся в массиве VBA.

Использование элемента управления `ListBox` для выделения строк на листе

Пример, приведенный в этом разделе, поможет вам в решении различных практических задач. Он позволяет отображать элемент управления `ListBox`, который состоит из элементов заполненного диапазона на текущем листе (рис. 14.16). Пользователь может выбрать несколько опций списка в элементе управления `ListBox`. Щелкнув на кнопке Все, вы выберете все опции, а щелкнув на кнопке Сброс, отмените выбор всех опций. Щелчок на кнопке OK приводит к выделению строк, которые соответствуют выделенным опциям элемента управления `ListBox`. Конечно, можно выделить несколько несмежных диапазонов непосредственно на листе. Эта задача выполняется с помощью клавиши <Ctrl>. Но со временем становится понятно, что метод, предложенный в этом разделе, намного удобнее.

Компакт-диск

Рассматриваемый в этом разделе пример находится на прилагаемом к книге компакт-диске в файле `listbox select rows.xlsm`.

Выбор нескольких опций возможен, поскольку свойству `MultiSelect` элемента управления `ListBox` присвоено значение `1` — `fmMultiSelectMulti`. Установка свойства `ListStyle` элемента управления `ListBox` равным `1` (`fmListStyleOption`) приводит к отображению флажков для каждого элемента.

Ниже приведена процедура объекта `UserForm`, применяемая для обработки события `Initialize`. Эта процедура создает объект `rng`, который состоит из используемого диапазона активного листа. Дополнительный код устанавливает свойства `RowSource` и `ColumnCount` элемента управления `ListBox`, а также изменяет значение свойства `ColumnWidths`, чтобы столбцы элемента управления `ListBox` по ширине соответствовали столбцам активного рабочего листа.

```
Private Sub UserForm_Initialize()
    Dim ColCnt As Integer
    Dim rng As Range
    Dim cw As String
```

```
   Dim c As Integer
   ColCnt = ActiveSheet.UsedRange.Columns.Count
   Set rng = ActiveSheet.UsedRange
   With ListBox1
      .ColumnCount = ColCnt
      .RowSource = rng.Address
      cw = ""
      For c = 1 To .ColumnCount
         cw = cw & rng.Columns(c).Width & ";"
      Next c
      .ColumnWidths = cw
      .ListIndex = 0
   End With
End Sub
```

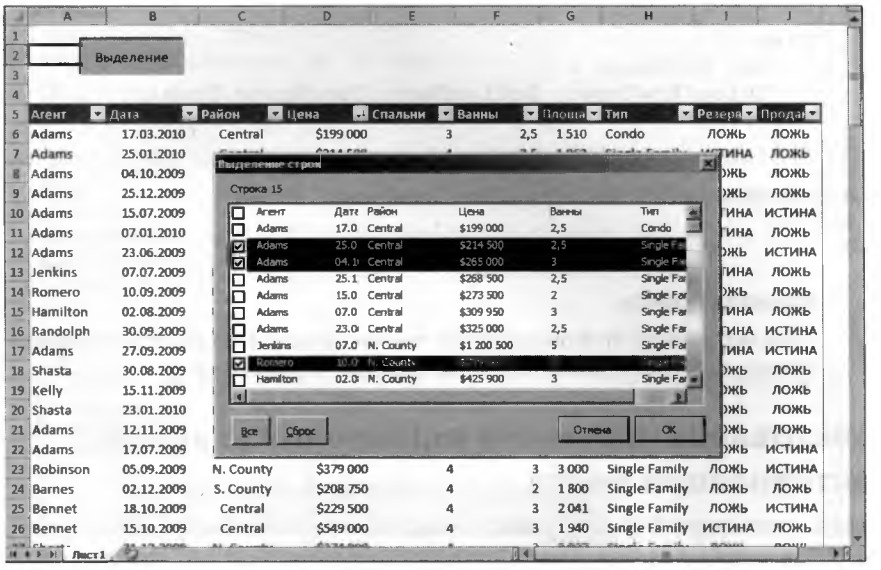


Рис. 14.16. Этот элемент управления `ListBox` облегчает выделение строк в рабочем листе

Кнопки **Все** и **Сброс** (называющиеся `SelectAllButton` и `SelectNoneButton`) имеют простые процедуры обработки событий, которые показаны ниже.

```
Private Sub SelectAllButton_Click()
   Dim r As Integer
   For r = 0 To ListBox1.ListCount - 1
      ListBox1.Selected(r) = True
   Next r
End Sub

Private Sub SelectNoneButton_Click()
   Dim r As Integer
   For r = 0 To ListBox1.ListCount - 1
      ListBox1.Selected(r) = False
   Next r
End Sub
```

Далее приведена процедура обработки события OKButton_Click. Эта процедура создает объект Range, называющийся RowRange. Он состоит из строк, соответствующих выделенным опциям в элементе управления ListBox. Для того чтобы определить факт выделения опции, в коде проверяется значение свойства Selected элемента управления ListBox. Обратите внимание на использование функции Union для добавления дополнительных диапазонов к объекту RowRange.

```
Private Sub OKButton_Click()
   Dim RowRange As Range
   RowCnt = 0
   For r = 0 To ListBox1.ListCount - 1
      If ListBox1.Selected(r) Then
         RowCnt = RowCnt + 1
         If RowCnt = 1 Then
            Set RowRange = ActiveSheet.UsedRange.Rows(r + 1)
         Else
            Set RowRange = _
            Union(RowRange, ActiveSheet.UsedRange.Rows(r + 1))
         End If
      End If
   Next r
   If Not RowRange Is Nothing Then RowRange.Select
   Unload Me
End Sub
```

Компакт-диск

Рассмотренный в этом разделе пример находится на прилагаемом к книге компакт-диске в файле listbox select rows.xlsm.

Использование элемента управления ListBox для активизации листа

Пример, приведенный в этой главе, и полезен, и познавателен. В нем использован элемент управления ListBox с несколькими столбцами (еще его называют многоколоночным) для отображения списка рабочих листов активной рабочей книги. В столбцах содержатся следующие данные:

- имя листа;
- тип листа (рабочий лист, диаграмма или диалоговый лист Excel 5/95);
- количество непустых ячеек в листе;
- состояние листа.

На рис. 14.17 показан пример подобного диалогового окна.

Код процедуры UserForm_Initialize (который приведен ниже) создает двумерный массив и собирает информацию, циклически просматривая листы активной рабочей книги. После этого массив передается в элемент управления ListBox.

```
Public OriginalSheet As Object
Private Sub UserForm_Initialize()
   Dim SheetData() As String
   Set OriginalSheet = ActiveSheet
   ShtCnt = ActiveWorkbook.Sheets.Count
   ReDim SheetData(1 To ShtCnt, 1 To 4)
   ShtNum = 1
```

```
  For Each Sht In ActiveWorkbook.Sheets
     If Sht.Name = ActiveSheet.Name Then _
        ListPos = ShtNum - 1
     SheetData(ShtNum, 1) = Sht.Name
     Select Case TypeName(Sht)
        Case "Worksheet"
           SheetData(ShtNum, 2) = "Лист"
           SheetData(ShtNum, 3) = _
              Application.CountA(Sht.Cells)
        Case "Chart"
           SheetData(ShtNum, 2) = "Диаграмма"
           SheetData(ShtNum, 3) = "Н/Д"
        Case "DialogSheet"
           SheetData(ShtNum, 2) = "Диалог"
           SheetData(ShtNum, 3) = "Н/Д"
     End Select
     If Sht.Visible Then
        SheetData(ShtNum, 4) = "Да"
     Else
        SheetData(ShtNum, 4) = "Нет"
     End If
     ShtNum = ShtNum + 1
  Next Sht
  With ListBox1
     .ColumnWidths = "100 pt;30 pt;40 pt;50 pt"
     .List = SheetData
     .ListIndex = ListPos
  End With
End Sub
```

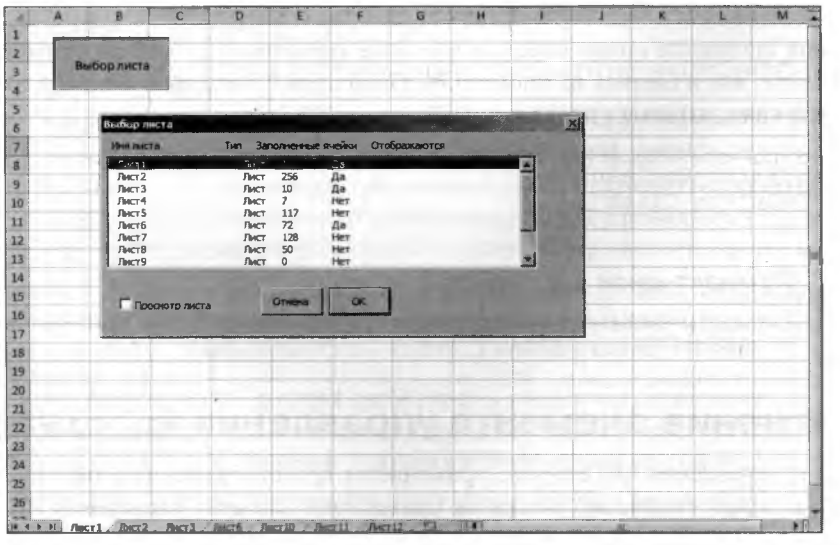


Рис. 14.17. С помощью этого диалогового окна пользователь активизирует лист

Ниже приводится код процедуры `ListBox1_Click`.

```
Private Sub ListBox1_Click()
   If cbPreview Then _
      Sheets(ListBox1.Value).Activate
End Sub
```

Значение элемента управления CheckBox (с названием cbPreview) определяет необходимость предварительного просмотра листа после того, как пользователь щелкнет на соответствующей опции списка элемента управления ListBox.

Щелчок на кнопке OK (объект OKButton) приводит к выполнению процедуры OKButton_Click, которая показана ниже.

```
Private Sub OKButton_Click()
   Dim UserSheet As Object
   Set UserSheet = Sheets(ListBox1.Value)
   If UserSheet.Visible Then
      UserSheet.Activate
   Else
      If MsgBox("Отобразить листы?", _
         vbQuestion + vbYesNoCancel) = vbYes Then
         UserSheet.Visible = True
         UserSheet.Activate
      Else
         OriginalSheet.Activate
      End If
   End If
   Unload Me
End Sub
```

Процедура OKButton_Click создает объектную переменную, которая представляет выделенный лист. Если лист отображается, то он активизируется. Если лист скрыт, то на экран выводится сообщение, в котором предлагается сделать лист видимым. Если пользователь даст утвердительный ответ на запрос, то лист будет отображен и активизируется. В противном случае активизируется исходный лист (который хранится в переменной OriginalSheet).

Двойной щелчок на опции списка в элементе управления ListBox приводит к тому же результату, что и щелчок на кнопке OK. Процедура ListBox1_DblClick, которая отображена ниже, вызывает процедуру OKButton_Click.

```
Private Sub ListBox1_DblClick(ByVal Cancel As _
   SForms.ReturnBoolean)
   Call OKButton_Click
End Sub
```

Компакт-диск

Рассматриваемый в этом разделе пример находится на прилагаемом к книге компакт-диске в файле listbox activate sheet.xlsm.

Применение элемента управления MultiPage

Элемент управления MultiPage применяется при отображении в пользовательских диалоговых окнах множества элементов управления. Элемент управления MultiPage позволяет группировать опции, а также размещать каждую группу на отдельной вкладке.

На рис. 14.18 показан пример формы UserForm, которая включает MultiPage. В этом случае элемент управления содержит три страницы, каждая из которых находится на своей вкладке.

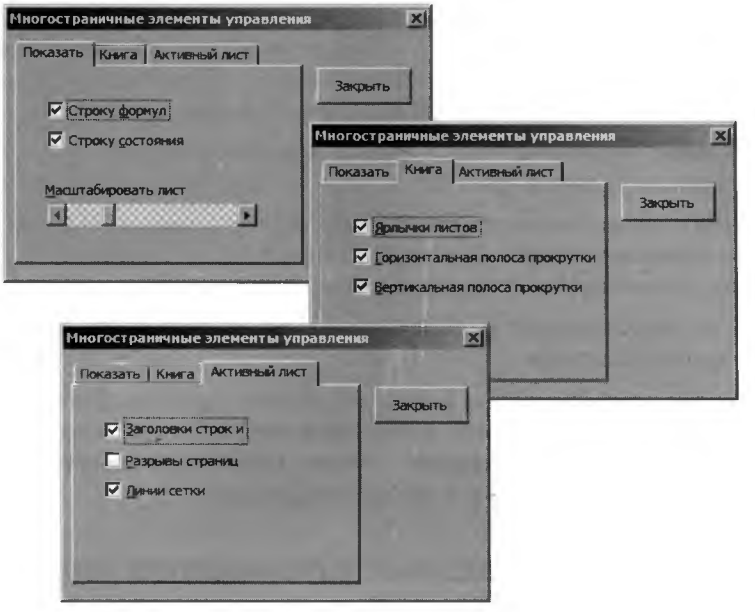


Рис. 14.18. Элемент управления `MultiPage` группирует элементы управления на страницах, доступ к которым обеспечивается с вкладки

Компакт-диск

Пример из этого раздела находится на прилагаемом к книге компакт-диске в файле `multipage control demo.xlsm`.

Примечание

Панель инструментов `Toolbox` также включает элемент управления `TabStrip`, напоминающий элемент управления `MultiPage`. Однако в отличие от `MultiPage`, элемент управления `TabStrip` не может включать другие объекты. Поскольку элемент управления `MultiPage` является более гибким, вряд ли вам придется обращаться к элементу управления `TabStrip`.

Применение элемента управления `MultiPage` может вызвать определенные проблемы. Ниже описаны факторы, которые необходимо учитывать при использовании элемента управления `MultiPage`.

- Вкладка (или страница), которая отображается поверх всех остальных, определяется значением свойства `Value` элемента управления `MultiPage`. Значение `0` соответствует первой вкладке. Значение `1` вызывает отображение второй вкладки и т.д.
- По умолчанию элемент управления `MultiPage` состоит из двух страниц. Для того чтобы добавить дополнительные вкладки, щелкните на любой вкладке правой кнопкой мыши и в контекстном меню выберите пункт New Page (Новая страница).
- При работе с элементом управления `MultiPage` щелкните на вкладке, чтобы установить свойства страницы. В окне Properties отобразятся свойства, значения которых можно изменить.
- Иногда сложно выделить сам элемент управления `MultiPage`, так как щелчок на нем приводит к выделению страницы элемента управления. Для того чтобы выде-

лить только элемент управления, щелкните на его границе. Кроме того, можете воспользоваться клавишей <Tab> для циклического перемещения между элементами управления. Еще одним вариантом выделения элемента управления является выбор пункта MultiPage из раскрывающегося списка окна Properties.

- Если элемент управления `MultiPage` содержит много вкладок, то присвойте его свойству `MultiRow` значение `True`, чтобы отобразить вкладки в несколько строк.
- Если необходимо, то вместо вкладок можно отображать кнопки. Достаточно изменить значение свойства `Style` на `1`. Если значение свойства `Style` равно `2`, элемент управления `MultiPage` не будет отображать ни вкладки, ни кнопки.
- Свойство `TabOrientation` определяет расположение вкладок на элементе управления `MultiPage`.
- Для каждой страницы можно установить эффект перехода. Для этого воспользуйтесь свойством `TransitionEffect`. Например, щелчок на вкладке приведет к тому, что новая страница "отодвинет" старую. Применяйте свойство `TransitionPeriod`, чтобы задать скорость эффекта перехода.

Использование внешних элементов управления

Пример, рассматриваемый в этом разделе, основан на элементе управления Microsoft Windows Media Player. Несмотря на то что он не является элементом управления Excel (настраивается при установке Windows), он прекрасно работает с формами `UserForm`.

Для того чтобы воспользоваться элементом управления Microsoft Windows Media Player выполните следующие действия.

1. Активизируйте среду VBE.
2. Щелкните правой кнопкой мыши на панели `Toolbox` и в контекстном меню выберите параметр Additional Controls (Дополнительные элементы управления).

   Если окно Toolbox не отображается, выполните команду View⇨Toolbox (Вид⇨ Окно Tookbox).
3. В окне Additional Controls установите флажок Windows Media Player.
4. Щелкните на кнопке OK.

   Набор инструментов пополнится новым элементом управления.

На рис. 14.19 показаны элемент управления Windows Media Player, встроенный в форму `UserForm`, а также окно Properties. Свойство `URL` определяет URL-ссылку воспроизводимой композиции (музыкальная запись или видеоролик). Если композиция находится на жестком диске вашего компьютера, свойство `URL` определяет полный путь и имя соответствующего файла.

На рис. 14.20 показан пример использования этого элемента управления. Для сокрытия окна, предназначенного для отображения видеороликов, была уменьшена высота окна элемента управления Windows Media Player. Также был добавлен список, созданный на основе элемента управления `ListBox`, в котором отображаются аудиофайлы MP3. После щелчка на кнопке Пуск (Play) начинается воспроизведение выбранного файла. Щелчок на кнопке Закрыть (Close) приведет к прекращению воспроизведения и к закрытию окна `UserForm`. Форма `UserForm` отображается в немодальном режиме, поэтому можно продолжать работу во время отображения диалогового окна.

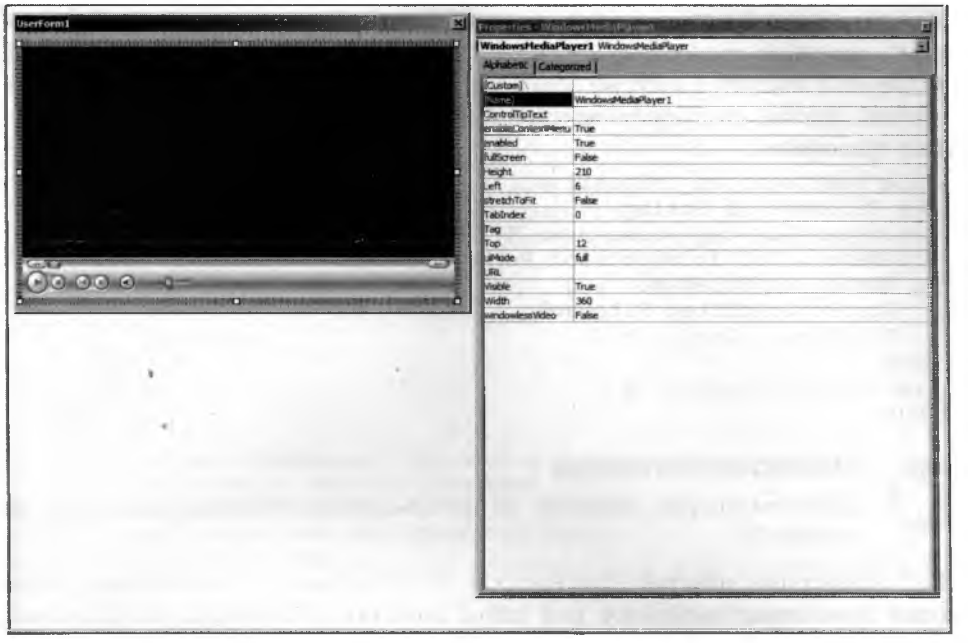


Рис. 14.19. Элемент управления Windows Media Player, встроенный в форму

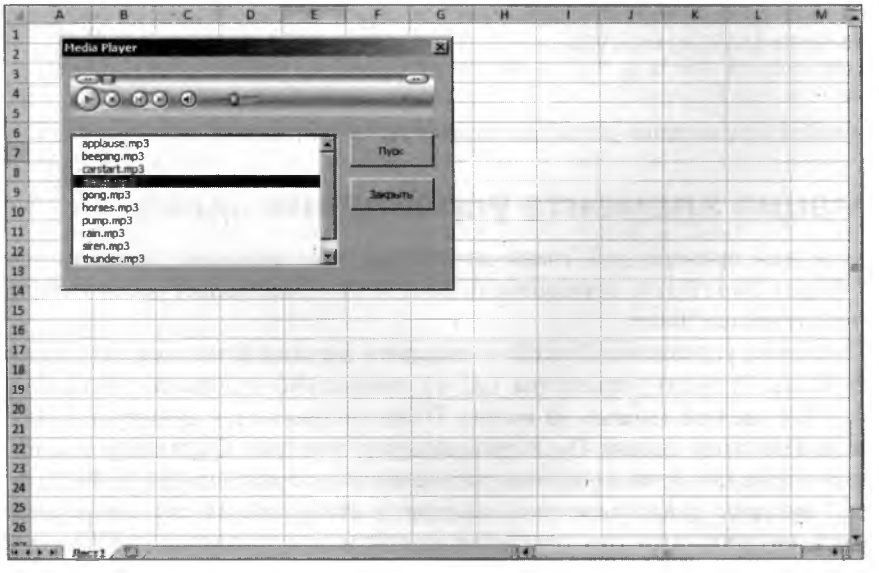


Рис. 14.20. Элемент управления Windows Media Player в действии

Компакт-диск

Рассматриваемый в этом разделе пример можно найти на прилагаемом к книге компакт-диске в файле `mediaplayer.xlsm`. Этот файл находится в отдельной папке вместе с бесплатно распространяемыми MP3-файлами.

Рассматриваемый пример весьма прост. Названия MP3-файлов в окне списка отображаются с помощью процедуры UserForm_Initialize. В целях упрощения алгоритма аудофайлы находятся в той же папке, что и рабочая книга. Можно реализовать и более гибкий подход, предусматривающий выбор пользователем папки, содержащей требуемые аудиофайлы.

```
Private Sub UserForm_Initialize()
   Dim FileName As String
'  Заполнение списка названиями MP3-файлов
   FileName = Dir(ThisWorkbook.Path & "\*.mp3", vbNormal)
   Do While Len(FileName) > 0
      ListBox1.AddItem FileName
      FileName = Dir()
   Loop
   ListBox1.ListIndex = 0
End Sub
```

Перекрестная ссылка

Дополнительные сведения об использовании команды Dir можно найти в главе 27.

Код обработчика событий PlayButton_Click включает единственный оператор, который присваивает выбранное имя файла свойству URL объекта WindowsMediaPlayer1.

```
Private Sub PlayButton_Click()
'  Свойство URL загружает музыкальный трек и запускает плеер
   WindowsMediaPlayer1.URL = _
   ThisWorkbook.Path & "\" & ListBox1.List(ListBox1.ListIndex)
End Sub
```

Подумайте о возможной доработке этого простого приложения.

Анимация элемента управления Label

В последнем примере этой главы демонстрируется анимация элемента управления Label. Форма UserForm, показанная на рис. 14.21, представляет собой интерактивный генератор случайных чисел.

Два элемента управления TextBox содержат верхнее и нижнее значения для случайного числа. Элемент управления Label изначально отображает четыре знака вопроса, а после щелчка мышью на кнопке Пуск отображаются анимированные случайные числа. При этом кнопка Пуск превращается в кнопку Остановить, а щелчок на ней мышью приводит к прекращению анимации и отображению случайного числа. На рис. 14.22 показано диалоговое окно, в котором отображается случайное число в диапазоне от 1 до 10000.

Ниже приводится код, связанный с кнопкой генератора случайных чисел.

```
Dim Stopped As Boolean
Private Sub StartStopButton_Click()
   Dim Low As Double, Hi As Double

   If StartStopButton.Caption = "Start" Then
'     Проверка минимального и максимального значений
      If Not IsNumeric(TextBox1.Text) Then
         MsgBox "Нечисловое значение.", vbInformation
```

```
        With TextBox1
            .SelStart = 0
            .SelLength = Len(.Text)
            .SetFocus
        End With
        Exit Sub
    End If

    If Not IsNumeric(TextBox2.Text) Then
        MsgBox "Нечисловое конечное значение.", vbInformation
        With TextBox2
            .SelStart = 0
            .SelLength = Len(.Text)
            .SetFocus
        End With
        Exit Sub
    End If

'   А теперь удостоверимся, что выбран правильный порядок
    Low = Application.Min(Val(TextBox1.Text), _
        Val(TextBox2.Text))
    Hi = Application.Max(Val(TextBox1.Text), _
        Val(TextBox2.Text))

'   Настройка размера шрифта (при необходимости)
    Select Case Application.Max(Len(TextBox1.Text), _
        Len(TextBox2.Text))
    Case Is < 5: Label1.Font.Size = 72
        Case 5: Label1.Font.Size = 60
        Case 6: Label1.Font.Size = 48
        Case Else: Label1.Font.Size = 36
    End Select

    StartStopButton.Caption = "Остановить"
    Stopped = False
    Randomize
    Do Until Stopped
        Label1.Caption = Int((Hi - Low + 1) * Rnd + Low)
        DoEvents ' Выполнение анимации
    Loop
  Else
    Stopped = True
    StartStopButton.Caption = "Пуск"
  End If
End Sub
```

Поскольку кнопка выполняет две функции (запуск и остановка анимации), в процедуре используется общедоступная переменная `Stopped`, с помощью которой отслеживается состояние кнопки. Первая часть процедуры состоит из двух структур `If-Then`, с помощью которых проверяется содержимое элементов управления `TextBox`. Два других подобных оператора позволяют удостовериться в том, что меньшая величина действительно не превосходит большей величины. В следующем разделе кода осуществляется настройка размера шрифта элемента управления `Label` на основании максимальной величины. Цикл `Do Until loop` генерирует и отображает случайные числа. Обратите внимание на оператор `DoEvents`. Он позволяет Excel использовать все возможности операционной системы. Если бы его не было, элемент управления `Label` не смог бы отобразить каждое генерируемое случайное число. Другими словами, именно оператор `DoEvents` делает возможной анимацию.

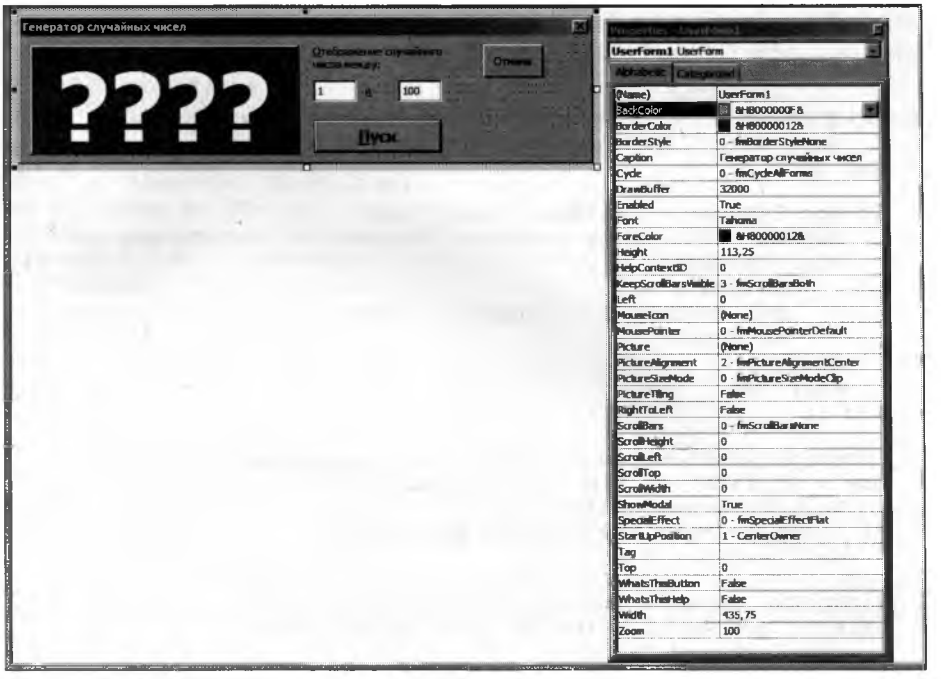


Рис. 14.21. Генератор случайных чисел

Рис. 14.22. Сгенерированное случайное число

Форма `UserForm` также включает элемент управления `CommandButton`, который выполняет функции кнопки **Отмена**. Этот элемент управления находится за пределами окна `UserForm`, поэтому невидим. Свойству `Cancel` элемента управления `CommandButton` присвоено значение `True`. Вследствие этого нажатие клавиши <Esc> дает тот же эффект, что и щелчок на кнопке **Отмена**. Соответствующая процедура обработки событий присваивает переменной `Stopped` значение `True` и выгружает форму `UserForm`.

```
Private Sub CancelButton_Click()
   Stopped = True
   Unload Me
End Sub
```

Компакт-диск

Пример из этого раздела находится на прилагаемом к книге компакт-диске в файле `random number generator.xlsm`.

Глава 15

Дополнительные приемы работы с пользовательскими формами

В этой главе...

- Немодальные диалоговые окна
- Отображение индикатора текущего состояния
- Создание мастеров
- Имитация работы функции `MsgBox`
- Диалоговое окно `UserForm` с перемещаемыми элементами управления
- Диалоговое окно `UserForm` без строки заголовка
- Имитация панели инструментов с помощью диалогового окна `UserForm`
- Диалоговое окно `UserForm` с изменяемыми размерами
- Несколько кнопок с одной процедурой обработки событий
- Диалоговое окно выбора цвета
- Отображение диаграммы в пользовательском диалоговом окне `UserForm`
- Создание полупрозрачной формы ввода данных
- Расширенная форма ввода данных
- Игра в "пятнашки"
- Играем в видеопокер в окне `UserForm`

В этой главе рассматриваются примеры, которые не вошли в главу 14, в частности дополнительные примеры форм `UserForm`.

Немодальные диалоговые окна

Большинство диалоговых окон, о которых речь шла выше, *модальные*, т.е. их необходимо удалять с экрана, прежде чем приступать к работе с окном приложения, находящимся под этим окном. Некоторые же диалоговые окна являются *немодальными*. Это означает, что пользователь может продолжать работу в приложении, даже когда диалоговое окно отображено на экране.

Для отображения немодального окна `UserForm` воспользуйтесь следующим оператором:

```
UserForm1.Show vbModeless
```

Слово `vbModeless` является встроенной константой, которая имеет значение `0`. Таким образом, представленный ниже оператор будет идентичен предыдущему.

```
UserForm1.Show 0
```

На рис. 15.1 показано немодальное диалоговое окно, которое отображает информацию об активной ячейке. Если диалоговое окно представлено на экране, пользователь может продолжать работу с ячейками, перемещаться на другие листы, а также выполнять другие действия, поддерживаемые в Excel.

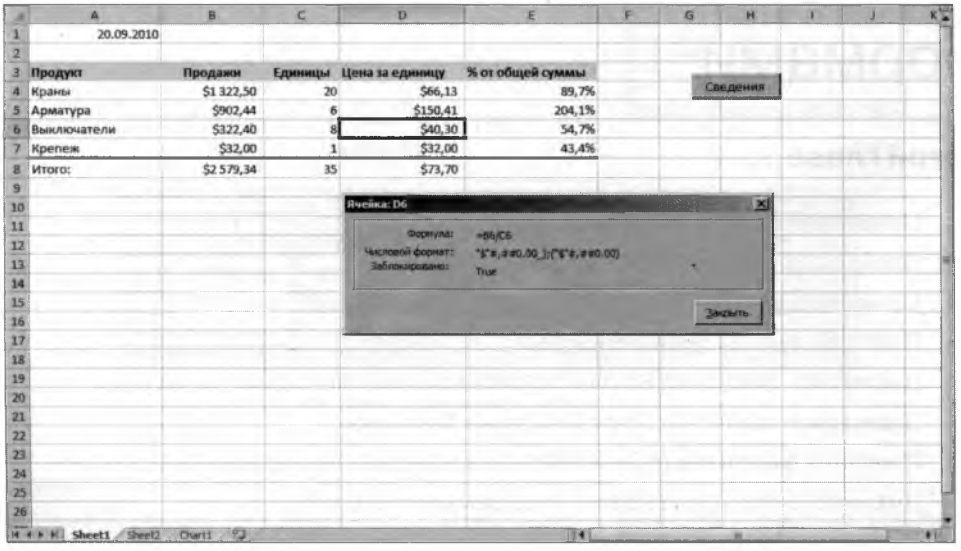


Рис. 15.1. Немодальное диалоговое окно остается видимым, даже если пользователь продолжает работать с электронной таблицей

Компакт-диск

Пример из данного раздела находится на прилагаемом к книге компакт-диске в файле `modeless userform1.xlsm`.

Важным моментом в использовании немодального диалогового окна является определение времени, когда необходимо обновить его содержимое. С этой целью в нашем примере используются два события рабочей книги: `SheetSelectionChange` и `SheetActivate`. Процедуры обработки этих событий находятся в модуле кода объекта `ThisWorkbook` (ЭтаКнига).

Перекрестная ссылка

Обратитесь к главе 19 за дополнительными сведениями о событиях.

Ниже приведен код процедур-обработчиков событий.

```
Private Sub Workbook_SheetSelectionChange _
   (ByVal Sh As Object, ByVal Target As Range)
   Call UpdateBox
End Sub

Private Sub Workbook_SheetActivate(ByVal Sh As Object)
   Call UpdateBox
End Sub
```

Описанные выше процедуры вызывают процедуру `UpdateBox`, код которой приводится ниже.

```
Sub UpdateBox()
   With UserForm1
'     Проверка активности листа
      If TypeName(ActiveSheet) <> "Worksheet" Then
         .lblFormula.Caption = "Н/Д"
         .lblNumFormat.Caption = "Н/Д"
         .lblLocked.Caption = "Н/Д"
         Exit Sub
      End If


      .Caption = "Ячейка: " & ActiveCell.Address(False, False)
'     Формула
      If ActiveCell.HasFormula Then
         .lblFormula.Caption = ActiveCell.Formula
      Else
         .lblFormula.Caption = "(нет)"
      End If

'     Числовой формат
      .lblNumFormat.Caption = ActiveCell.NumberFormat
'      Заблокировано
      .lblLocked.Caption = ActiveCell.Locked
   End With
End Sub
```

Процедура `UpdateBox` изменяет заголовок диалогового окна `UserForm`, который отображает адрес активной ячейки. После этого обновляются три элемента управления `Label` (`lblFormula`, `lblNumFormat` и `lblLocked`).

Ниже приведена информация, которая поможет понять, как работает этот код.

- Форма `UserForm` отображается в немодальном режиме, вследствие чего она остается на экране, даже если пользователь работает с листом.
- Код в верхней части процедуры проверяет, является ли рабочий лист активным. Если это не рабочий лист, то элементы управления `Label` получают заголовок `Н/Д`.
- Активная ячейка отслеживается благодаря событию `Selection_Change` (которое обрабатывается в модуле `ThisWorkbook` (ЭтаКнига)).
- Информация отображается в элементе управления `Label` пользовательской формы.

На рис. 15.2 показан более сложный пример. Данная версия отображает достаточно много дополнительной информации о выделенной ячейке. Пользователи, которые давно работают с Excel, могут заметить, что это диалоговое окно напоминает диалоговое окно Инфо (Info) (оно было удалено из Excel несколько лет назад). Код этого примера слишком велик для того, чтобы приводить его в книге, но вы сможете его увидеть (вместе с дополнительными комментариями) в примере рабочей книги.

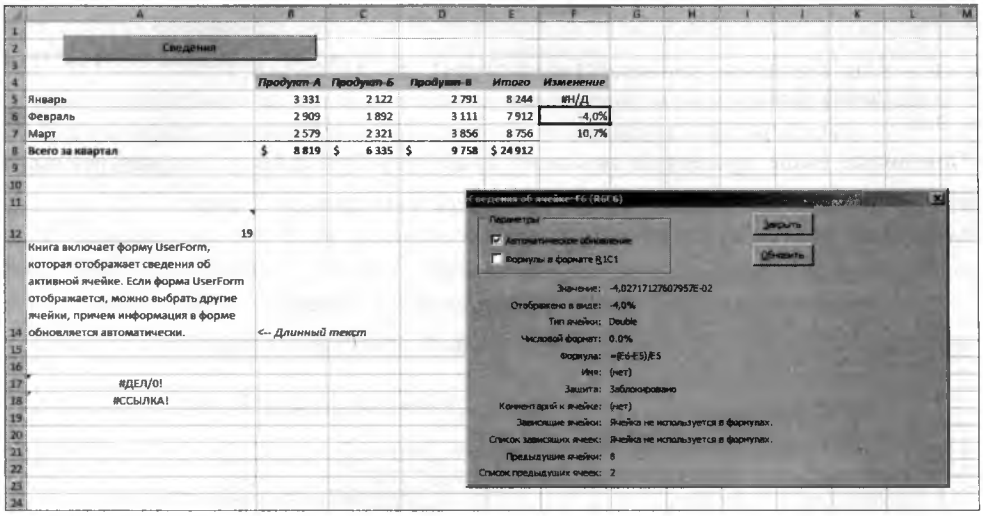


Рис. 15.2. В этом немодальном окне `UserForm` отображаются сведения об активной ячейке

Компакт-диск

Этот пример под названием `modeless userform2.xlsm` находится на прилагаемом к книге компакт-диске.

А теперь приведем несколько замечаний, касающихся этого более сложного примера.

- Диалоговое окно `UserForm` содержит флажок Автоматическое обновление (Auto Update). Если этот флажок установлен, диалоговое окно обновляется автоматически. Если же он не установлен, для обновления информации нужно щелкнуть мышью на кнопке Обновить (Update).
- Рабочая книга использует модуль класса с целью обнаружения двух событий для всех открытых рабочих книг: `SheetSelectionChange` и `SheetActivate`. В результате код, отображающий информацию о текущей ячейке, выполняется автоматически независимо от того, в какой рабочей книге происходят события (предполагается, что установлен флажок Автоматическое обновление). Некоторые действия (например, изменение числового формата ячейки) не приводят к вызову какого-либо из описанных событий. Поэтому диалоговое окно `UserForm` также содержит кнопку Обновить.

Перекрестная ссылка

Обратитесь к главе 29 для получения дополнительных сведений о модулях классов.

- Счетчики зависящих ячеек и ячеек, от которых зависит текущая, отображают данные только для активного листа. Это ограничение свойств `Precedents` и `Dependents`.
- Так как длина отображаемой информации может изменяться, код VBA изменяет размер и расстояние между элементами управления `Label`, а также размер самого диалогового окна `UserForm` в соответствии с длиной отображаемой информации.

Отображение индикатора текущего состояния

Одним из самых волнующих вопросов для разработчиков приложений Excel является возможность использования *индикатора текущего состояния*. Это графический "измеритель", который отображает текущее состояние выполняемой задачи, например долго работающего макроса. Отобразить на экране подобный индикатор сравнительно несложно.

В этом разделе рассматриваются методы создания индикаторов текущего состояния.

- Макрос, который запускается за пределами диалогового окна `UserForm` (отдельный индикатор текущего состояния).
- Макрос, который запускается из диалогового окна `UserForm`. В этом случае диалоговое окно `UserForm` использует элемент управления `MultiPage` для отображения индикатора текущего состояния, пока выполняется другой макрос.
- Макрос, который запускается из диалогового окна `UserForm`. В этом случае высота диалогового окна `UserForm` увеличивается, а индикатор текущего состояния отображается в нижней части окна.

При использовании индикатора текущего состояния необходимо знать, насколько завершено текущее задание. Способы получения этой информации различаются в зависимости от типа выполняемого макроса. Например, если макрос записывает данные в ячейки (и количество таких ячеек известно), то остается создать код, который будет подсчитывать процентное отношение количества ячеек, содержащих данные. Даже если невозможно точно оценить, насколько далеко "зашел" макрос, пользователю небезынтересно будет узнать, что макрос еще выполняется и Excel "пребывает в добром здравии".

Отображение индикатора текущего состояния в строке состояния окна

Простой способ отображения хода выполнения макроса — использование строки состояния Excel. Его преимущество — простота реализации. Недостатком же является то, что большинство пользователей не привыкли отслеживать информацию, которая отображается в строке состояния окна, поскольку предпочитают просматривать ее в отдельном окне.

Для отображения сообщения в строке состояния используется следующий оператор:

```
Application.StatusBar = "Пожалуйста, подождите..."
```

Можно обновлять строку состояния в процессе выполнения макроса. Например, если в макросе используется переменная `Pct`, которая представляет состояние задачи, можно создать код, который будет периодически выполнять следующий оператор:

```
Application.StatusBar = "Выполнение... " & Pct & "% завершено"
```

После завершения макроса нужно вернуть строку состояния к прежнему виду. Для этого используется следующий оператор:

```
Application.StatusBar = False
```

Если строка состояния не возвращена к прежнему виду, продолжает отображаться завершающее сообщение.

Предупреждение

Помните о том, что индикатор текущего состояния замедляет выполнение макроса, поскольку обновление индикатора требует дополнительного использования системных ресурсов. Если быстродействие макроса превыше всего, от использования индикатора текущего состояния лучше отказаться.

Создание отдельного индикатора текущего состояния

В этом разделе описывается процесс создания отдельного индикатора текущего состояния для отображения процесса выполнения макроса. Такой индикатор не инициализируется путем отображения формы `UserForm`. Этот макрос просто очищает рабочий лист, а также записывает 20 тысяч случайных чисел в диапазон ячеек.

```
Sub GenerateRandomNumbers()
'  Вставка случайных чисел в активный рабочий лист
   Const RowMax As Integer = 500
   Const ColMax As Integer = 40
   Dim r As Integer, c As Integer
   If TypeName(ActiveSheet) <> "Worksheet" Then Exit Sub
   Cells.Clear
   For r = 1 To RowMax
      For c = 1 To ColMax
         Cells(r, c) = Int(Rnd * 1000)
      Next c
      Next r
End Sub
```

После небольшого изменения макроса (описанного ниже) диалоговое окно `UserForm`, показанное на рис. 15.3, отображает индикатор процесса выполнения макроса.

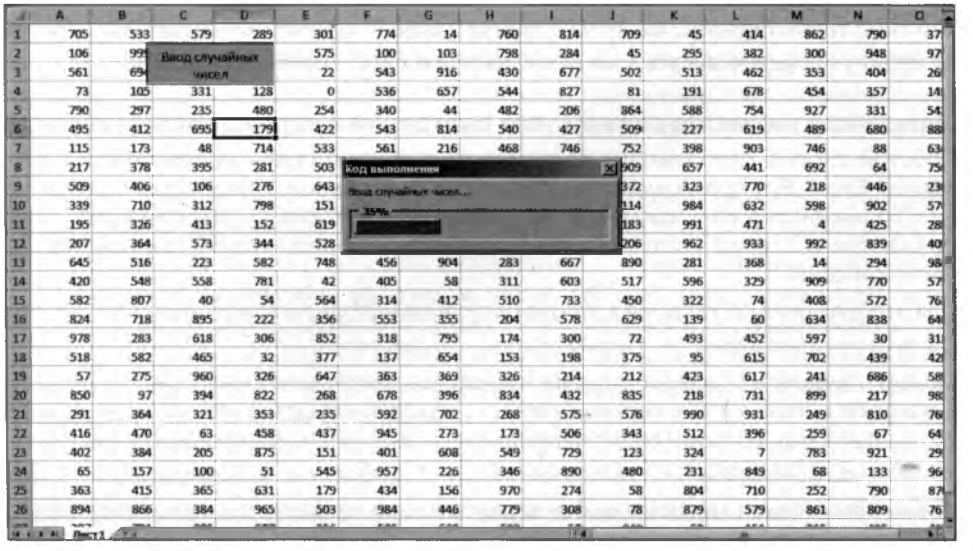


Рис. 15.3. В окне `UserForm` отображается ход выполнения макроса

Компакт-диск

Этот пример находится на прилагаемом к книге компакт-диске в файле `progress indicator1.xlsm`.

Создание диалогового окна `UserForm`, включающего индикатор текущего состояния

Ниже приведены инструкции по созданию диалогового окна `UserForm`, которое будет отображать текущее состояние выполняемой задачи.

1. Вставьте новое диалоговое окно `UserForm` и измените значение свойства `Caption` на `Ход выполнения процесса`.
2. Добавьте элемент управления `Frame` и присвойте ему имя `FrameProgress`.
3. Добавьте элемент управления `Label` в состав элемента управления `Frame` и назначьте ему имя `LabelProgress`. Удалите заголовок этого элемента управления, а также сделайте его фон красным (посредством свойства `BackColor`). На данный момент размеры и расположение этого элемента управления не важны.
4. Добавьте еще один элемент управления `Label` над элементом управления `Frame`, с помощью которого вы будете описывать происходящее (необязательно). В нашем примере с помощью этого элемента управления добавляется надпись `Ход выполнения процесса`.
5. Настройте диалоговое окно `UserForm` и элементы управления таким образом, чтобы они выглядели, как на рис. 15.4.

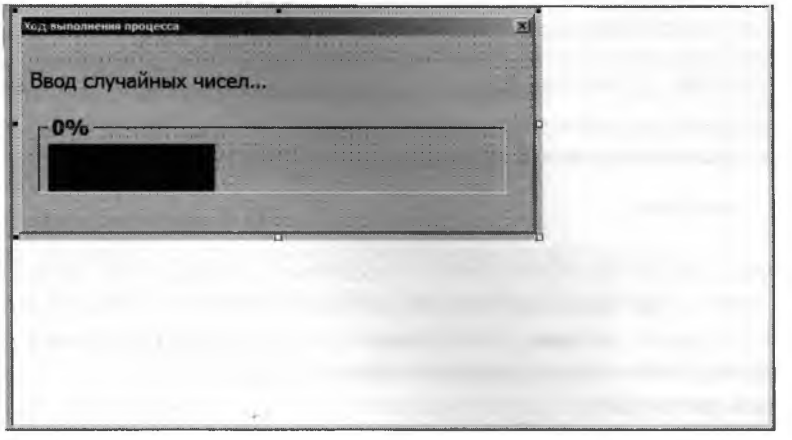


Рис. 15.4. Окно формы `UserForm` может играть роль индикатора хода выполнения процесса

Можно изменить тип форматирования элементов управления. Например, я изменил свойство `SpecialEffect` элемента `Frame` таким образом, что последний стал "вдавленным".

Создание процедур обработки событий

В данном случае важно, чтобы процедура автоматически запускалась при отображении диалогового окна `UserForm`. Один из вариантов подразумевает использование события `Initialize`. Но это событие возникает еще до того, как диалоговое окно ото-

бражается на экране, поэтому такой вариант не подходит. С другой стороны, событие Activate возникает в тот момент, когда диалоговое окно UserForm отображается на экране, поэтому в данном случае можно остановиться на его использовании.

Вставьте приведенную ниже процедуру в модуль кода диалогового окна UserForm. Эта процедура всего лишь вызывает процедуру GenerateRandomNumbers, когда диалоговое окно UserForm отображается на экране. Процедура GenerateRandomNumbers, которая хранится в модуле кода VBA, является фактическим макросом, который будет работать, пока на экране отображается индикатор текущего состояния.

```
Private Sub UserForm_Activate()
   Call GenerateRandomNumbers
End Sub
```

Ниже представлен код процедуры GenerateRandomNumber. Обратите внимание на наличие дополнительного модуля, который отслеживает текущее состояние, сохраняя соответствующие сведения в переменной PctDone.

```
Sub GenerateRandomNumbers()
'  Вставка случайных чисел в активный рабочий лист
   Dim Counter As Integer
   Const RowMax As Integer = 500
   Const ColMax As Integer = 40
   Dim r As Integer, c As Integer
   Dim PctDone As Single

   If TypeName(ActiveSheet) <> "Worksheet" Then Exit Sub
   Cells.Clear
   Counter = 1
   For r = 1 To RowMax
      For c = 1 To ColMax
         Cells(r, c) = Int(Rnd * 1000)
         Counter = Counter + 1
      Next c
      PctDone = Counter / (RowMax * ColMax)
      Call UpdateProgress(PctDone)
   Next r
   Unload UserForm1
End Sub
```

Процедура GenerateRandomNumbers включает два цикла. Во внутреннем цикле вызывается процедура UpdateProgress, которая принимает только один аргумент (переменная PctDone, которая "ответственна" за отображение процесса выполнения макроса). Эта переменная может принимать значения от 0 до 100.

```
Sub UpdateProgress(Pct)
   With UserForm1
      .FrameProgress.Caption = Format(Pct, "0%")
      .LabelProgress.Width = Pct * (.FrameProgress.Width - 10)
      .Repaint
   End With
End Sub
```

Создание процедуры запуска

Все, что осталось сделать, — написать процедуру отображения диалогового окна UserForm. Введите следующий код в модуль VBA.

```
Sub ShowUserForm()
   With UserForm1
```

```
    .LabelProgress.Width = 0
    .Show
  End With
End Sub
```

Совет

Можно сделать так, чтобы строка состояния соответствовала текущей теме рабочей книги. Для этого в процедуру `ShowUserForm` добавьте следующий оператор.

```
.LabelProgress.BackColor = ActiveWorkbook.Theme. _
   ThemeColorScheme.Colors(msoThemeAccent1)
```

Принцип действия процедуры запуска

При выполнении процедуры `ShowUserForm` ширина объекта `Label` устанавливается равной `0`. После этого вызывается метод `Show` объекта `UserForm1`, что приводит к отображению диалогового окна `UserForm` (которое играет роль индикатора текущего состояния). Когда диалоговое окно `UserForm` отображается на экране, вызывается событие `Activate`, которое приводит к выполнению процедуры `GenerateRandomNumbers`. Процедура `GenerateRandomNumbers` включает код, который вызывает процедуру `UpdateProgress` при каждом изменении переменной счетчика цикла `r`. Обратите внимание, что процедура `UpdateProgress` использует метод `Repaint` объекта `UserForm`. Если бы этого оператора не было, изображение на экране не обновлялось бы. Перед завершением процедуры `GenerateRandomNumbers` ее последний оператор выгружает диалоговое окно `UserForm` из памяти.

Для того чтобы модифицировать эту методику, необходимо разобраться, как определяется процент завершения выполняемой задачи. После этого значение состояния задачи можно будет присваивать переменной `PctDone`. Определение этой величины осуществляется различными способами в зависимости от приложения. Если код выполняется в цикле (как в данном примере), вычисление процента выполнения сравнительно несложное. Если же код выполняется не в цикле, определение состояния выполнения происходит в различных точках кода.

Отображение сведений о текущем состоянии с помощью элемента управления `MultiPage`

В предыдущем примере макрос запускался не из диалогового окна `UserForm`. Во многих случаях долго выполняющийся макрос можно вынудить "уйти со сцены", щелкнув мышью на кнопке OK в диалоговом окне `UserForm`. В этом же разделе будет описан лучший способ, использование которого возможно при следующих допущениях:

- проект завершен и отлажен;
- в проекте используется диалоговое окно `UserForm` (без элемента управления `MultiPage`) для запуска долго выполняющегося макроса;
- существует метод оценки степени завершения выполняемой задачи.

Компакт-диск

Пример, демонстрирующий описанный прием, находится на прилагаемом к книге компакт-диске в файле `progress indicator2.xlsm`.

Как и в предыдущем примере, в рабочий лист вводятся случайные числа. Отличие заключается в том, что в приложении содержится диалоговое окно `UserForm`, в котором пользователем определяется количество строк и столбцов для ввода случайных чисел (рис. 15.5).

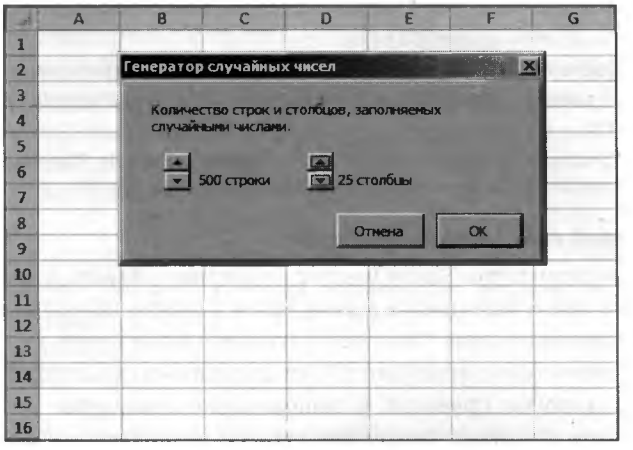


Рис. 15.5. Пользователь определяет количество строк и столбцов, в которые вводятся случайные числа

Изменение диалогового окна

На данном этапе предполагается, что диалоговое окно `UserForm` уже настроено. Нам осталось добавить элемент управления `MultiPage`. Первая страница элемента управления `MultiPage` будет содержать все первоначальные элементы управления. На второй странице располагаются элементы управления, которые используются для отображения индикатора текущего состояния. Когда макрос начнет выполняться, в коде VBA значение свойства `Value` элемента управления `MultiPage` изменится. В результате будут скрыты исходные элементы управления и отображены элементы управления, которые используются для создания индикатора текущего состояния.

Первым шагом будет добавление элемента управления `MultiPage` в диалоговое окно `UserForm`. После этого необходимо переместить все существующие в диалоговом окне `UserForm` элементы управления на первую страницу элемента управления `MultiPage`.

Затем следует активизировать вторую страницу элемента управления `MultiPage` и настроить ее так, чтобы она выглядела, как показано на рис. 15.6. В данном случае используется та же комбинация элементов управления, что и в предыдущем примере.

1. Добавьте элемент управления `Frame`, присвоив ему имя `FrameProgress`.
2. Добавьте элемент управления `Label` в состав элемента управления `Frame`, присвоив ему имя `LabelProgress`. Удалите заголовок этого элемента управления, а также сделайте его фоновый цвет красным.
3. Добавьте еще один элемент управления `Label`, описывающий суть происходящего (необязательно).
4. Активизируйте элемент управления `MultiPage` в целом (а не отдельную вкладку), а свойству `Style` присвойте значение `2` — `fmTabStyleNone`.

   (Это приведет к сокрытию всех вкладок.) Возможно, придется изменить размер элемента управления `MultiPage`, чтобы учесть скрытые вкладки.

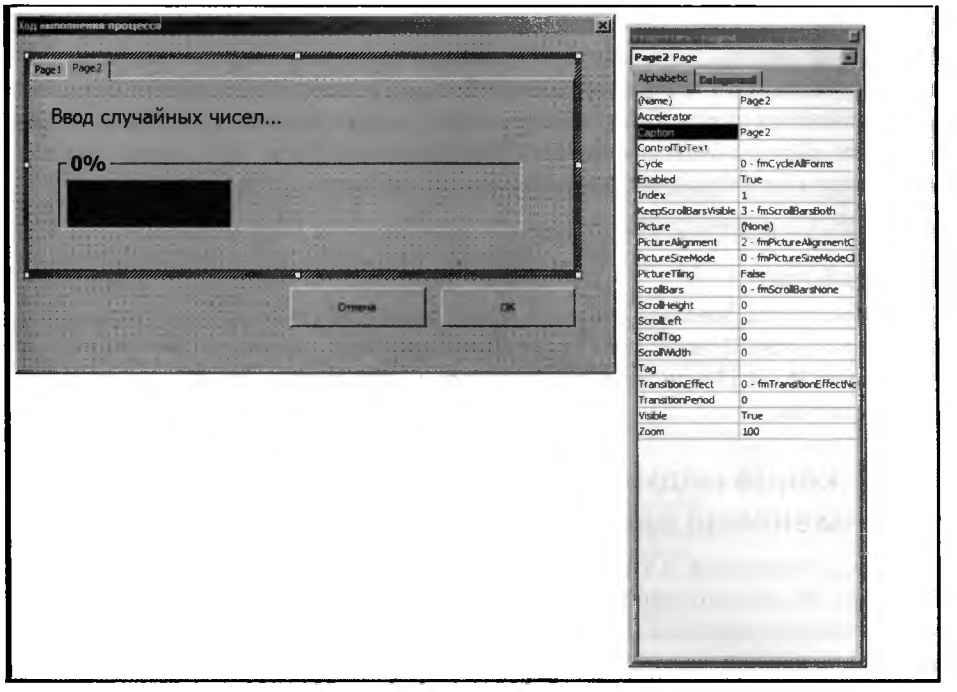


Рис. 15.6. Вторая вкладка элемента управления `MultiPage`, которая применяется для отображения индикатора текущего состояния

Совет

Простейший способ выделения элемента управления `MultiPage`, когда вкладки скрыты, — выбор его в раскрывающемся списке, который находится в окне Properties. Для выбора определенной страницы укажите величину свойства `Value` для элемента `MultiPage`: `0` — для `Page1`, `1` — для `Page2` и т.д.

Вставка процедуры `UpdateProgress`

Вставьте следующую процедуру в модуль кода диалогового окна `UserForm`.

```
Sub UpdateProgress(Pct)
  With UserForm1
     .FrameProgress.Caption = Format(Pct, "0%")
     .LabelProgress.Width = Pct * (.FrameProgress.Width - 10)
     .Repaint
  End With
End Sub
```

Процедура `UpdateProgress` вызывается из макроса после щелчка пользователем на кнопке OK и выполняет обновление индикатора текущего состояния.

Изменение процедуры

Далее необходимо модифицировать процедуру, которая выполняется после щелчка на кнопке OK. Данная процедура выступает обработчиком события `Click` и называется `OKButton_Click`. Для начала необходимо вставить оператор в начало процедуры.

```
MultiPage1.Value = 1
```

Этот оператор приводит к активизации второй вкладки элемента управления MultiPage (страницы, на которой отображается индикатор текущего состояния).

На следующем шаге необходимо самостоятельно создать код, который будет вычислять степень выполнения задачи. Полученное значение следует присвоить переменной PctDone. Скорее всего, расчеты будут производиться внутри цикла. После этого нужно добавить приведенный ниже оператор, который будет обновлять индикатор текущего состояния.

```
Call UpdateProgress(PctDone)
```

Как это работает

Данная методика довольно проста и, кроме того, требует использования только одного диалогового окна UserForm. В коде программа переходит к другой странице элемента управления MultiPage и превращает нормальное диалоговое окно в индикатор текущего состояния.

Отображение индикатора текущего состояния без применения элемента управления MultiPage

Пример, приведенный в этом разделе, подобен методу, представленному в предыдущем разделе. Но рассмотренная далее методика немного проще, так как не требует применения элемента управления MultiPage. Вместо этого индикатор текущего состояния содержится в нижней части диалогового окна UserForm. Для того чтобы элементы управления, составляющие индикатор текущего состояния, изначально не были видны, высота диалогового окна UserForm была уменьшена до соответствующего размера. Как только нужно будет отобразить индикатор текущего состояния, высота диалогового окна UserForm увеличится, что сделает индикатор видимым на экране.

Компакт-диск

Пример описанной методики находится на прилагаемом к книге компакт-диске в файле progress indicator3.xlsm.

На рис. 15.7 показано диалоговое окно UserForm в редакторе VBE. Свойство Height диалогового окна UserForm имеет значение 172. Но перед тем как отобразить диалоговое окно UserForm, значение свойства Height устанавливается равным 124 (это приводит к тому, что элементы управления, представляющие индикатор текущего состояния, становятся невидимыми). Как только пользователь щелкнет на кнопке OK, код VBA изменит значение свойства Height на 172. Для этого используется следующий оператор:

```
Me.Height = 172
```

На рис. 15.8 показано диалоговое окно UserForm с отображенным индикатором текущего состояния.

Создание мастеров

Во многих приложениях для предоставления пользователям пошаговых инструкций по выполнению определенных задач используются специальные мастера. Мастер импорта текстовых файлов Excel является хорошим примером такого подхода к решению задач. *Мастер* — это, по сути, последовательность диалоговых окон, которые предоставляют пользователю информацию и запрашивают у него необходимые сведения. Часто

выбор пользователя в первых диалоговых окнах влияет на содержимое последующих окон. Как правило, пользователю предоставляется возможность свободно перемещаться вперед и назад по последовательности диалоговых окон. Кроме того, он может щелкнуть на кнопке Готово (Finish), чтобы использовать значения, принятые по умолчанию.

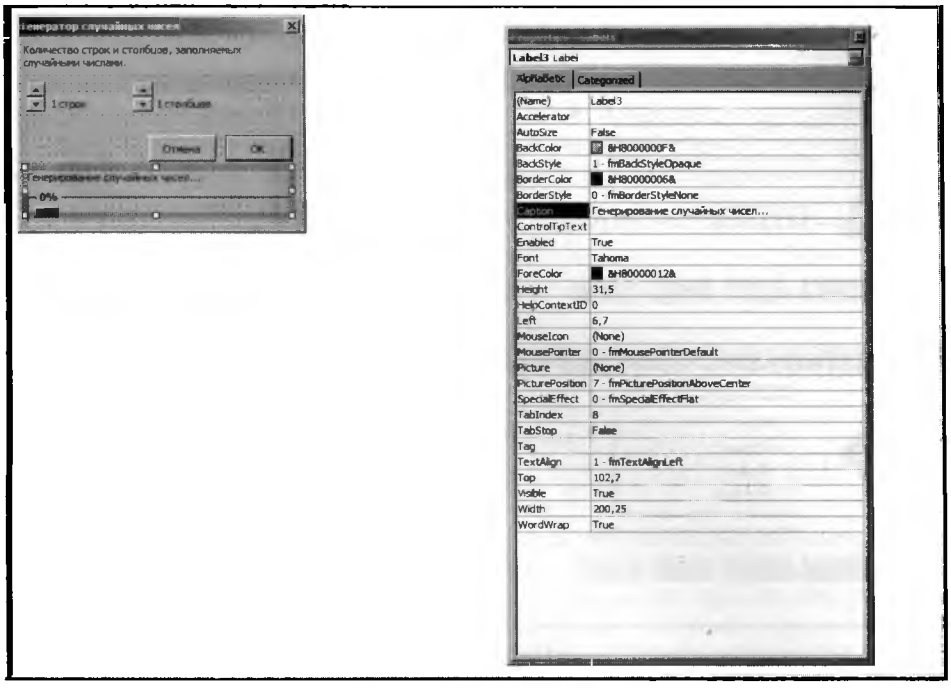


Рис. 15.7. Уменьшение высоты диалогового окна `UserForm` приводит к сокрытию элементов управления, образующих индикатор текущего состояния

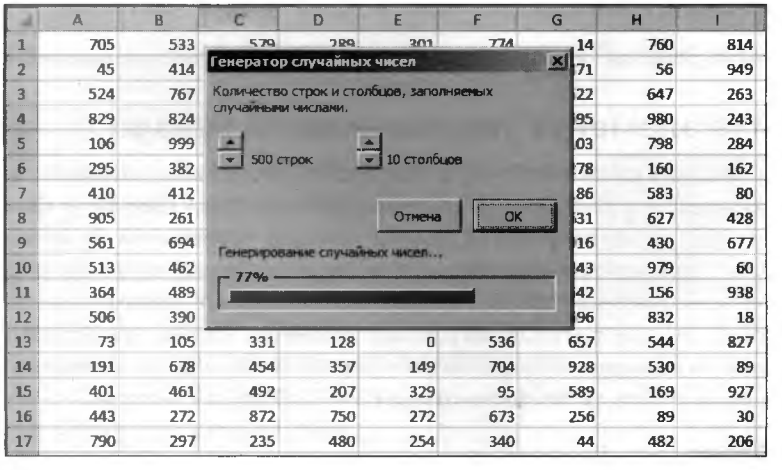


Рис. 15.8. Индикатор текущего состояния в действии

Вы можете создать "мастер" посредством VBA-кода и использования последовательности диалоговых окон `UserForm`. Однако существует более эффективный способ соз-

дания мастера с помощью единственного диалогового окна `UserForm` и элемента управления `MultiPage` со скрытыми вкладками.

На рис. 15.9 показан пример простого мастера, который включает четыре этапа. Этот мастер состоит из диалогового окна `UserForm`, в котором содержится элемент управления `MultiPage`. Каждый этап работы мастера соответствует отдельной вкладке элемента управления `MultiPage`.

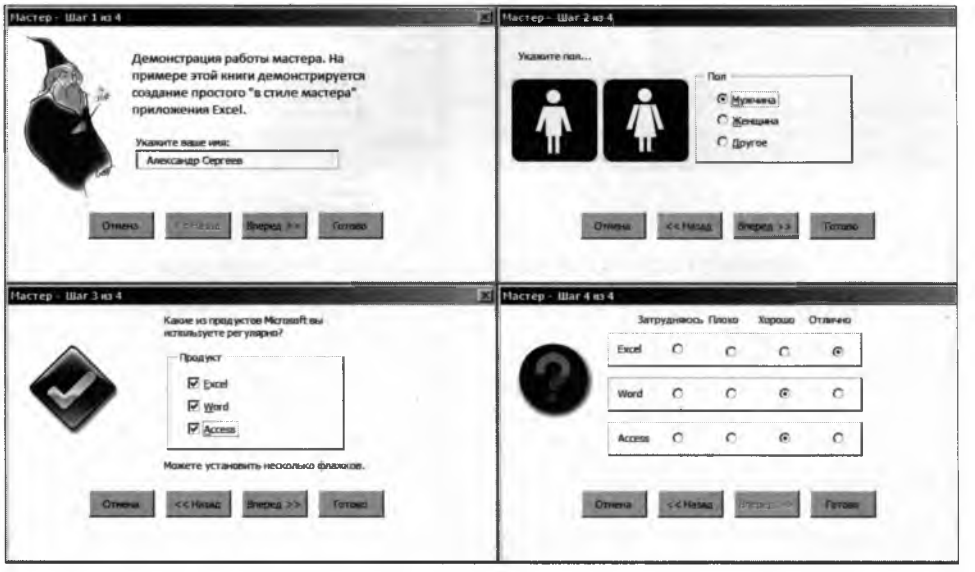


Рис. 15.9. Состоящий из четырех этапов мастер, использующий элемент управления `MultiPage`

Компакт-диск

Пример мастера, рассмотренный в следующем разделе, находится на прилагаемом к книге компакт-диске в файле `wizard demo.xlsm`.

В следующих разделах описывается создание полноценного приложения-мастера.

Настройка элемента управления `MultiPage`

Начните с создания диалогового окна `UserForm`. После этого добавьте элемент управления `MultiPage`. По умолчанию он содержит две страницы. Щелкните правой кнопкой мыши на элементе управления `MultiPage` и вставьте достаточное количество страниц, которые будут использоваться при создании мастера (по одной странице на каждый этап работы мастера). Пример мастера на компакт-диске включает четыре этапа, поэтому элемент управления `MultiPage` содержит четыре страницы. Имена страниц элемента управления `MultiPage` в данном случае роли не играют. Свойство `Style` элемента управления `MultiPage` должно быть установлено равным 2 — `fmTabStyleNone`.

Совет

При настройке диалогового окна `UserForm` вкладки (страницы) необходимо сделать видимыми (чтобы иметь возможность обращаться к разным страницам элемента управления `MultiPage`).

Добавьте все необходимые элементы управления на каждую страницу элемента управления MultiPage. Эти элементы будут зависеть от целей конкретного приложения. Вы также вправе изменить размер элемента управления MultiPage, чтобы обеспечить место для всех элементов управления.

Добавление кнопок

Далее необходимо добавить кнопки, которые будут управлять переходом между этапами работы мастера. Эти кнопки должны размещаться за пределами элемента управления MultiPage, поскольку они используются при отображении любой страницы элемента управления MultiPage. Как правило, мастера имеют четыре стандартные кнопки.

- **Отмена.** Отменяет работу мастера, а также результаты всех ранее выполненных действий.
- **Назад.** Позволяет перейти к предыдущему этапу работы мастера. На первом этапе эта кнопка должна быть неактивной.
- **Вперед.** Позволяет перейти к следующему этапу работы мастера. На последнем этапе эта кнопка должна быть неактивной.
- **Готово.** Позволяет завершить работу мастера.

Примечание

В одних случаях пользователь может щелкнуть на кнопке Готово на любом этапе выполнения мастера, в результате чего будут использованы значения, принятые по умолчанию. В других случаях мастер требует ответа пользователя на некоторые вопросы. Если возникает подобная ситуация, кнопка Готово должна быть отключена до тех пор, пока пользователь не введет все необходимые сведения. В примере, находящемся на прилагаемом к книге компакт-диске, требуется ввести информацию в текстовое поле на первом этапе работы мастера.

В рассматриваемом примере элементы управления CommandButton (кнопки) в коде называются CancelButton, BackButton, NextButton и FinishButton.

Программирование кнопок

В каждой из четырех кнопок мастера необходима процедура обработки события Click. Ниже приведена процедура обработки события для кнопки CancelButton. Она использует функцию MsgBox (рис. 15.10), чтобы проверить, завершена ли работа мастера. Если пользователь щелкнет на кнопке Отмена, то диалоговое окно UserForm будет выгружено из памяти и никакие действия выполнены не будут. Этот тип проверки не является обязательным.

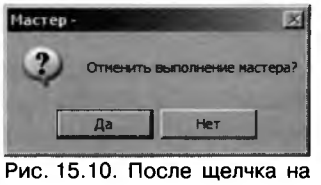


Рис. 15.10. После щелчка на кнопке Отмена отображается соответствующее сообщение

```
Private Sub CancelButton_Click()
  Dim Msg As String
  Dim Ans As Integer
  Msg = "Отменить мастер?"
  Ans = MsgBox(Msg, vbQuestion + vbYesNo, APPNAME)
```

```
   If Ans = vbYes Then Unload Me
End Sub
```

Ниже приведена процедура обработки событий для кнопок Назад и Вперед.

```
Private Sub BackButton_Click()
   MultiPage1.Value = MultiPage1.Value - 1
   UpdateControls
End Sub
Private Sub NextButton_Click()
   MultiPage1.Value = MultiPage1.Value + 1
   UpdateControls
End Sub
```

Эти две процедуры очень просты. Они изменяют значение свойства `Value` элемента управления `MultiPage`, после чего вызывают другую процедуру, которая называется `UpdateControls` (данная процедура будет показана ниже).

Процедура `UpdateControls` отвечает за включение и отключение кнопок `BackButton` и `NextButton`.

```
Sub UpdateControls()
   Select Case MultiPage1.Value
      Case 0
         BackButton.Enabled = False
         NextButton.Enabled = True
      Case MultiPage1.Pages.Count - 1
         BackButton.Enabled = True
         NextButton.Enabled = False
      Case Else
         BackButton.Enabled = True
         NextButton.Enabled = True
      End Select

'  Обновить заголовок
   Me.Caption = APPNAME & " Шаг " _
      & MultiPage1.Value + 1 & " из " _
      & MultiPage1.Pages.Count
'  Поле Имя заполнять обязательно
   If tbName.Text = "" Then
      FinishButton.Enabled = False
   Else
      FinishButton.Enabled = True
   End If
End Sub
```

Процедура изменяет заголовок диалогового окна `UserForm`, и в результате он отображает текущий этап работы мастера и общее количество этапов (константа `APPNAME` является глобальной и определена в модуле кода `Module1`). После этого проверяется содержимое поля Имя на первой странице элемента управления `MultiPage` (для создания этого поля используется элемент управления `TextBox`, который называется `tbName`). Данное поле обязательно нужно заполнить, поэтому кнопка Готово отключена до момента заполнения. Если элемент управления `TextBox` остается пустым, кнопка `FinishButton` отключена. В противном случае кнопка Готово активизируется, и у пользователя появляется возможность щелкнуть на ней.

Программирование зависимостей

В большинстве мастеров ответ пользователя на определенном этапе может повлиять на элементы управления, которые отображаются на последующих этапах. В примере на прилагаемом к книге компакт-диске на третьем этапе пользователь должен указать, какие программы он применяет в своей работе. После этого (на четвертом этапе) пользователю предлагается оценить выбранные программные продукты Microsoft. Элемент управления `OptionButton` для каждого продукта отображается только в том случае, если пользователь выбрал этот продукт на предыдущем этапе.

С точки зрения программирования эта задача реализуется в результате обработки события `Change` элемента управления `MultiPage`. Как только значение элемента управления `MultiPage` изменится (после щелчка на кнопке **Назад** или **Вперед**), будет запущена процедура `MultiPage1_Change`. Если в элементе управления активна последняя страница (четвертый этап), то процедура проверяет значения элементов управления `CheckBox` на странице, соответствующей третьему этапу работы мастера. После этого на странице для четвертого этапа выполняются необходимые изменения элементов управления.

В данном примере используются два массива: один — для элементов управления `CheckBox`, соответствующих продуктам (используется на третьем этапе), а второй — для элементов управления `Frame` (используется на четвертом этапе). Цикл `For Next` скрывает элементы управления `Frame` для тех продуктов, которые не были выбраны на предыдущем этапе. После этого изменяется вертикальное расположение отображаемых на экране элементов управления `Frame`. Если на странице, соответствующей третьему этапу работы мастера, не был выбран ни один из продуктов, то на последнем этапе скрываются все элементы управления, кроме `TextBox`, который содержит сообщение. Щелкните на кнопке **Готово** для выхода (если, конечно, на первом этапе введено имя). Процедура `MultiPage1_Change` приведена ниже.

```
Private Sub MultiPage1_Change()
'  Настраивать страницу рейтинга?
   If MultiPage1.Value = 3 Then
'     Создание массива элементов управления CheckBox
      Dim ProdCB(1 To 3) As MSForms.CheckBox
      Set ProdCB(1) = cbExcel
      Set ProdCB(2) = cbWord
      Set ProdCB(3) = cbAccess

'     Создание массива элементов управления Frame
      Dim ProdFrame(1 To 3) As MSForms.Frame
      Set ProdFrame(1) = FrameExcel
      Set ProdFrame(2) = FrameWord
      Set ProdFrame(3) = FrameAccess

      TopPos = 22
      FSpace = 8
      AtLeastOne = False
'     Просмотр всех продуктов
      For i = 1 To 3
         If ProdCB(i) Then
            ProdFrame(i).Visible = True
            ProdFrame(i).Top = TopPos
            TopPos = TopPos + ProdFrame(i).Height + Fspace
            AtLeastOne = True
         Else
```

```
            ProdFrame(i).Visible = False
         End If
      Next i

'     Ни один из продуктов не выбран?
      If AtLeastOne Then
         lblHeadings.Visible = True
         Image4.Visible = True
         lblFinishMsg.Visible = False
      Else
         lblHeadings.Visible = False
         Image4.Visible = False
         lblFinishMsg.Visible = True
         If tbName = "" Then
            lblFinishMsg.Caption = _
               "Введите имя на этапе 1."
         Else
            lblFinishMsg.Caption = _
               "Щелкните на кнопке Готово для выхода."
         End If
      End If
   End If
End Sub
```

Выполнение задачи

Когда пользователь щелкает на кнопке Готово, мастер выполняет свою задачу: перемещает информацию из диалогового окна `UserForm` в следующую пустую строку рабочего листа. Эта процедура под названием `FinishButton_Click` довольно проста. Она начинается с определения следующей пустой строки рабочего листа и задания значения переменной (`r`). Остальная часть процедуры выполняет идентификацию значений элементов управления и ввод данных в ячейки листа.

```
Private Sub FinishButton_Click()
   Dim r As Long
   r = Application.WorksheetFunction. _
      CountA(Range("A:A")) + 1
'  Вставить имя
   Cells(r, 1) = tbName.Text

'  Вставить пол
   Select Case True
      Case obMale: Cells(r, 2) = "Мужчина"
      Case obFemale: Cells(r, 2) = "Женщина"
      Case obNoAnswer: Cells(r, 2) = "Неизвестно"
   End Select
'  Определить используемость
   Cells(r, 3) = cbExcel
   Cells(r, 4) = cbWord
   Cells(r, 5) = cbAccess

'  Вставить оценки
   If obExcel1 Then Cells(r, 6) = ""
   If obExcel2 Then Cells(r, 6) = 0
   If obExcel3 Then Cells(r, 6) = 1
   If obExcel4 Then Cells(r, 6) = 2
   If obWord1 Then Cells(r, 7) = ""
```

```
    If obWord2 Then Cells(r, 7) = 0
    If obWord3 Then Cells(r, 7) = 1
    If obWord4 Then Cells(r, 7) = 2
    If obAccess1 Then Cells(r, 8) = ""
    If obAccess2 Then Cells(r, 8) = 0
    If obAccess3 Then Cells(r, 8) = 1
    If obAccess4 Then Cells(r, 8) = 2
'   Выгрузить форму
    Unload Me
End Sub
```

Как только мастер будет испытан и все станет работать должным образом, можно изменить значение свойства `Style` элемента управления `MultiPage`. Это свойство должно иметь значение 2 — `fmTabStyleNone`.

Имитация работы функции MsgBox

Функция VBA `MsgBox` достаточно необычна. В отличие от остальных функций она отображает диалоговое окно. С другой стороны, она, как и другие функции, возвращает значение — целое число, представляющее кнопку, на которой щелкнул пользователь.

В приведенном ниже примере содержится пользовательская функция, которая эмулирует функцию VBA `MsgBox`. Вначале может показаться, что это легкая задача. Учитывая то, что функция `MsgBox` невероятно гибкая (благодаря огромному количеству аргументов, которые она принимает), можно утверждать: создать пользовательскую функцию, эмулирующую поведение функции `MsgBox`, весьма непросто.

Примечание

Основная наша задача заключается не в создании альтернативной функции для отображения окон сообщений. Необходимо продемонстрировать разработку достаточно сложной функции, которая использует диалоговое окно `UserForm`. Кроме того, некоторым пользователям нравится сама идея создания функций для отображения окон сообщений. Созданную функцию легко модифицировать. Например, можно изменять используемый шрифт, цвет, надписи на кнопках и т.д.

Создаваемая функция, которая будет имитировать поведение функции `MsgBox`, получила название `MyMsgBox`. Имитация не является безупречной, так как функция `MyMsgBox` имеет следующие ограничения:

- не поддерживает аргумент *справка* (он способствует появлению кнопки **Справка**, щелчок на которой позволяет открыть файл справочной системы);
- не поддерживает аргумент *раздел* (он указывает раздел в файле справки);
- не поддерживает опцию *модальности* (которая приостанавливает работу всей программы до тех пор, пока пользователь не ответит на запрос окна сообщения);
- не воспроизводит звук при вызове.

Ниже приведен синтаксис функции `MyMsgBox`.

```
MyMsgBox( запрос[, кнопки] [, заголовок])
```

Этот синтаксис полностью соответствует синтаксису функции `MsgBox`, кроме того что в первой не поддерживаются два последних аргумента (*справка* и *раздел*). Функ-

ция MyMsgBox использует те же предопределенные константы, что и функция MsgBox: vbOKOnly, vbQuestion, vbDefaultButton1 и т.д.

Примечание

Если вы не знакомы с функцией VBA MsgBox, обратитесь к справочной системе для получения информации о ее аргументах.

Код функции MyMsgBox

Функция MyMsgBox использует диалоговое окно UserForm, которое называется MyMsgBoxForm. Данная функция очень короткая. Ее текст приводится ниже. Основная часть работы выполняется в функции обработки события Initialize (UserForm Initialize).

Компакт-диск

Полный код функции MyMsgBox слишком длинный, чтобы приводить его здесь, но он доступен в рабочей книге msgbox emulation.xlsm, находящейся на прилагаемом к книге компакт-диске. Эта рабочая книга настроена так, что вы сможете проверять работу различных опций.

```
Public Prompt1 As String
Public Buttons1 As IntegerPublic Title1 As String
Public UserClick As Integer
Function MyMsgBox(ByVal Prompt As String, _
   Optional ByVal Buttons As Integer, _
   Optional ByVal Title As String) As Integer
      Prompt1 = Prompt
      Buttons1 = Buttons
      Title1 = Title
   MyMsgBoxForm.Show
   MyMsgBox = UserClick
End Function
```

На рис. 15.11 показан результат выполнения функции MyMsgBox. Он напоминает окно сообщения VBA, но в этом случае применяется другой шрифт для выделения текста сообщения (Calibri, 12 пунктов, полужирный), а также используются другие пиктограммы.

Если к вашему компьютеру подключено несколько мониторов, форма UserForm может не попасть в центральную часть окна Excel. Для устранения этой проблемы воспользуйтесь следующим кодом, отображающим форму MyMsgBoxForm.

```
With MyMsgBoxForm
   .StartUpPosition = 0
   .Left = Application.Left + (0.5 * Application.Width) - _
      0.5 * .Width)
   .Top = Application.Top + (0.5 * Application.Height) - _
      (0.5 * .Height)
   .Show
End With
```

Ниже приведен код, который использовался для вызова функции на выполнение.

```
Prompt = "Вы выбрали сохранение рабочей книги " & vbCrLf
Prompt = Prompt & "на диске, который недоступен" & vbCrLf
Prompt = Prompt & "для всех сотрудников." & vbCrLf & vbCrLf
Prompt = Prompt & "Продолжать?"
Buttons = vbQuestion + vbYesNo
```

```
Title = "У нас проблема"
Ans = MyMsgBox(Prompt, Buttons, Title)
```

Примечание

Не волнуйтесь, результат выполнения этого примера вовсе не означает недоступность диска для пользователей вашего компьютера.

Как это работает

Обратите внимание на использование четырех переменных с глобальной областью действия. Первые три переменные (`Prompt1`, `Buttons1` и `Title1`) представляют аргументы, которые передаются функции. Еще одна переменная (`UserClick`) указывает значения, возвращаемые функцией. Процедура `UserForm_Initialize` нуждается в способе получения этой информации и отправке ее обратно в функцию. Использование глобальных переменных (`Public`) является единственной возможностью реализации необходимого механизма.

В диалоговом окне `UserForm` (рис. 15.12) есть четыре элемента управления `Image` (по одному для каждой возможной пиктограммы), три элемента `CommandButton`, а также элемент управления `TextBox`.

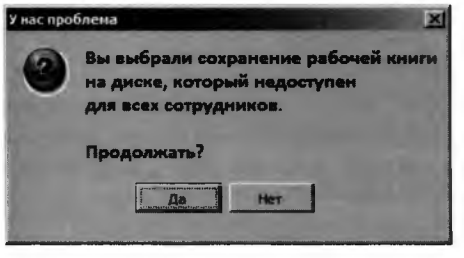


Рис. 15.11. Результат эмуляции работы функции `MsgBox`

Рис. 15.12. Диалоговое окно `UserForm`, которое используется в функции `MyMsgBox`

Примечание

Изначально для хранения четырех пиктограмм использовались элементы управления `Image`, в результате чего вокруг изображений появился едва заметный контур. Для получения отчетливого контура я воспользовался элементами управления `Label`.

Процедура `UserForm_Initialize` проверяет значения аргументов и выполняет следующие действия:

- определяет, какое изображение необходимо вывести на экран (остальные изображения скрываются);
- определяет, какие кнопки необходимо вывести на экран (остальные кнопки скрываются);
- определяет, какая кнопка является выбранной по умолчанию;
- выравнивает кнопки в диалоговом окне по центру;
- определяет подписи для элементов управления `CommandButton`;

- определяет расположение текста в диалоговом окне;
- определяет ширину диалогового окна (используется API-функция, которая предоставляет информацию о разрешении экрана);
- определяет необходимую высоту диалогового окна;
- отображает диалоговое окно `UserForm`.

Три дополнительные процедуры обработки событий используются для элементов управления `CommandButton`. Эти процедуры определяют, на какой из кнопок щелкнул пользователь. Затем переменной `UserClick` присваивается соответствующее значение.

Интерпретация второго аргумента (`Кнопки`) может оказаться немного сложнее. Этот аргумент может включать ряд констант и выглядеть следующим образом:

```
VbYesNoCancel + VbQuestion + VbDefaultButton3
```

Данный аргумент создает окно сообщения с тремя кнопками `MsgBox` (Да, Нет и Отмена), отображает пиктограмму со знаком вопроса и делает третью кнопку выбранной по умолчанию. Аргумент равен `547` (3+32+512). Основной трудностью в данном случае может оказаться получение трех фрагментов информации на основе одного числа. Решить данную проблему несложно: преобразуйте числа в двоичную форму и проверьте состояние отдельных битов этого аргумента. Например, число 547 в двоичной форме записывается как 1000100011. Двоичные цифры 4–6 определяют отображаемую пиктограмму, цифры 8–10 — отображаемые кнопки, а цифры 1 и 2 — кнопку, которая будет выбрана по умолчанию.

Использование функции `MyMsgBox`

Для того чтобы применить эту функцию в собственном проекте, экспортируйте модуль `MyMsgBoxMod` и диалоговое окно `MyMsgBoxForm`. Затем можете импортировать эти два файла в собственный проект. Используйте функцию `MyMsgBox` в коде точно так же, как функцию `MsgBox`.

Диалоговое окно `UserForm` с перемещаемыми элементами управления

Этот пример вряд ли имеет практическую ценность, но зато он помогает разобраться в событиях мыши. В диалоговом окне `UserForm`, показанном на рис. 15.13, содержатся три элемента управления `Image`. Пользователь может перетаскивать эти изображения в диалоговом окне.

Компакт-диск

Рассматриваемый в этой главе пример доступен на прилагаемом к книге компакт-диске и находится в файле `move controls.xlsm`.

Каждый из элементов управления `Image` включает две ассоциированные с ним процедуры обработки событий: `MouseDown` и `MouseMove`. Ниже приводится код процедур обработки событий для элемента управления `Image1` (другие процедуры идентичны, за исключением используемых имен элементов управления).

```
Private Sub Image1_MouseDown(ByVal Button As Integer, _
    ByVal Shift As Integer, ByVal X As Single, ByVal Y As Single)
'   Начальная позиция в момент щелчка на кнопке
```

```
   OldX = X
   OldY = Y
   Image1.ZOrder 0
End Sub

Private Sub Image1_MouseMove(ByVal Button As Integer, _
   ByVal Shift As Integer, ByVal X As Single, ByVal Y As Single)
' Перемещение изображения
   If Button = 1 Then
      Image1.Left = Image1.Left + (X - OldX)
      Image1.Top = Image1.Top + (Y - OldY)
   End If
End Sub
```

Рис. 15.13. Три элемента управления `Image` можно перетаскивать и сортировать с помощью мыши

После нажатия кнопки мыши происходит событие `MouseDown` и сохраняется информация о координатах X и Y указателя мыши. Для отслеживания исходных координат элементов управления используются две переменные с глобальной областью доступа — `OldX` и `OldY`. Эта процедура также изменяет свойство `ZOrder`, в результате чего изображение помещается "поверх" остальных изображений.

В процессе перемещения указателя мыши повторно вызывается событие `MouseMove`. Процедура обработки событий проверяет состояние кнопки мыши. Если аргумент `Button` равен `1`, значит, нажата левая кнопка мыши. В этом случае элемент управления `Image` смещен относительно своего исходного положения.

Также обратите внимание на то, как изменяется вид указателя мыши при его установке над изображением. Причина этого явления в том, что значение свойства `MousePointer` равно `15` — `fmMousePointerSizeAll`. Этот стиль указателя мыши обычно используется для того, чтобы показать, что объект перемещается.

Диалоговое окно `UserForm` без строки заголовка

В Excel существует возможность отображения диалогового окна `UserForm` без строки заголовка. Для этого используются некоторые API-функции. Пример подобного окна показан на рис. 15.14.

Еще один пример диалогового окна UserForm без строки заголовка показан на рис. 15.15. В нем находятся элементы управления Image и CommandButton.

Рис. 15.14. Это диалоговое окно UserForm лишено строки заголовка

Рис. 15.15. Еще одно диалоговое окно UserForm без строки заголовка

Компакт-диск

Оба эти примера находятся в рабочей книге no title bar.xlsm на прилагаемом компакт-диске. На этом компакт-диске находится еще одна версия примера заставки, рассматриваемого в главе 14. Она называется splash screen2.xlsm и отображает диалоговое окно UserForm без строки заголовка.

Для отображения диалогового окна UserForm без строки заголовка используются четыре функции Windows API: GetWindowLong, SetWindowLong, DrawMenuBar и FindWindowA (объявление этих функций можно увидеть в файле примера, находящемся на прилагаемом к книге компакт-диске). Эти функции вызываются процедурой UserForm_Initialize.

```
Private Sub UserForm_Initialize()
   Dim lngWindow As Long, 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
```

При работе с диалоговыми окнами без строки заголовка возникает проблема — невозможно перемещать их по экрану с помощью мыши. Эта проблема решается с помощью событий MouseDown и MouseMove, как описано в предыдущем разделе.

Примечание

Поскольку функция FindWindowA использует заголовок диалогового окна UserForm, этот прием не работает, если свойству Caption присвоена пустая строка.

Имитация панели инструментов с помощью диалогового окна UserForm

Создать настраиваемую панель инструментов в версиях, предшествующих Excel 2007, было относительно просто. С появлением Excel 2007 это стало невозможно. Точнее, остается возможность создания пользовательской панели инструментов с помощью VBA, но Excel игнорирует большинство инструкций VBA. В Excel 2007 все настраиваемые панели инструментов отображаются в группе ленты Надстройки⇨Настраиваемые панели инструментов (Add-Ins⇨Custom Toolbars). Эти панели инструментов нельзя перемещать, "отправлять в плавание", изменять их размеры либо закреплять.

В этом разделе вы найдете описание методики создания альтернативы панели инструментов: немодального диалогового окна UserForm, которое симулирует плавающую панель инструментов. На рис. 15.16 показано диалоговое окно UserForm, заменяющее панель инструментов.

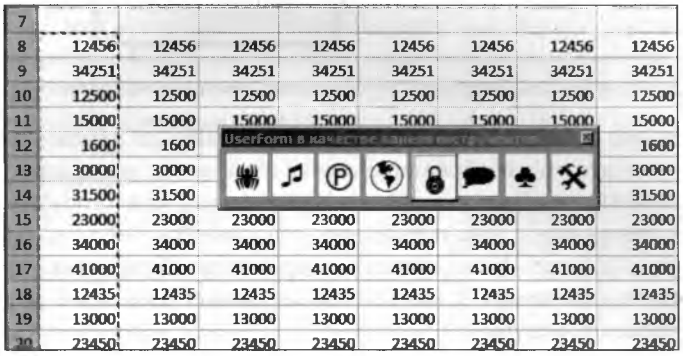


Рис. 15.16. Диалоговое окно UserForm, играющее роль панели инструментов

Компакт-диск

Этот пример находится на прилагаемом к книге компакт-диске в файле simulated toolbar.xlm.

В диалоговом окне UserForm содержится восемь элементов управления Image, каждый из которых вызывает макрос. На рис. 15.17 показано диалоговое окно UserForm, отображаемое в окне VBE. Обратите внимание на следующее:

- элементы управления не выровнены;
- размеры диалогового окна UserForm уменьшены;
- строка заголовка имеет стандартный размер.

Код VBA заботится о "косметических деталях" — выравнивает элементы управления, а также настраивает размеры диалогового окна UserForm таким образом, чтобы занять свободное пространство. Кроме того, код обращается к функциям Windows API, с помощью которых уменьшается размер строки заголовка окна UserForm — как на реальной панели инструментов. Для придания диалоговому окну UserForm еще большего сходства с панелью инструментов были установлены свойства ControlTipText для каждого элемента управления Image. Благодаря этому отображаются инструментальные

подсказки при установке указателя мыши поверх элемента управления, весьма напоминающие подсказки панелей инструментов.

Рис. 15.17. Диалоговое окно UserForm, имитирующее панель инструментов

Если вы откроете соответствующий файл, находящийся на прилагаемом к книге компакт-диске, то заметите, что изображение слегка изменяется после того, как над ним установлен указатель мыши. Причина этого в том, что для каждого элемента управления Image имеется связанный с ним обработчик событий MouseMove, который изменяет свойство SpecialEffect. Ниже приведена процедура обработчика событий MouseMove для элемента управления Image1 (для остальных элементов управления применяются аналогичные процедуры).

```
Private Sub Image1_MouseMove(ByVal Button As Integer, _
   ByVal Shift As Integer, ByVal X As Single, ByVal Y As Single)
      Call NoRaise
      Image1.SpecialEffect = fmSpecialEffectRaised
End Sub
```

Эта процедура вызывает процедуру NoRaise, которая отключает специальный эффект, вызванный из каждого элемента управления.

```
Private Sub NoRaise()
'  Удаляет эффект, вызванный из каждого элемента управления
   Dim ctl As Control
   For Each ctl In Controls
      ctl.SpecialEffect = fmSpecialEffectFlat
   Next ctl
End Sub
```

Полученный в этом случае эффект заключается в том, что формируется визуальная обратная связь с пользователем, устанавливающим указатель мыши поверх элемента управления (как на реальной панели инструментов). Но на этом сходство заканчивается, поскольку изменять размеры диалогового окна UserForm невозможно (например, отображать изображения в вертикальном направлении, а не в горизонтальном). Также невозможно пристыковать "псевдопанель" инструментов к одной из границ окна Excel.

Совет

Изображения, отображаемые на элементах управления, представляют собой символы из набора шрифтов Wingding. Для ввода символов в ячейки применялась команда Excel Вставка⇨Текст⇨Символ (Insert⇨Text⇨Symbol). Затем символы копировались в буфер обмена с последующей вставкой в свойство `Picture` в окне Properties. Это простой и быстрый способ добавления изображений в элементы управления окна `UserForm`.

Диалоговое окно `UserForm` с изменяемыми размерами

В Excel используется ряд диалоговых окон с изменяемыми размерами. Например, можно изменить размеры диалогового окна Диспетчер имен (Name Manager) путем щелчка и перетаскивания нижнего правого угла окна.

Если вы попытаетесь создать диалоговое окно с изменяемыми размерами, то очень быстро обнаружите, что нет прямого способа сделать это. Первое решение заключается в использовании вызовов Windows API. Этот метод работает, но сложен в реализации. В этом разделе представлен более простой метод, обеспечивающий создание диалогового окна `UserForm` с изменяемыми размерами.

Примечание

Этот прием был разработан Энди Поупом, экспертом по Excel и профессионалом Microsoft (MVP), проживающим в Великобритании. Энди — один из наиболее креативных разработчиков Excel. Желающие пообщаться с ним (а также загрузить интересные примеры) могут посетить его веб-сайт `http://andypope.info`.

На рис. 15.18 показано диалоговое окно `UserForm`, описанное в этом разделе. Оно включает элемент управления `ListBox`, отображающий данные из рабочего листа. Обратите внимание на полосы прокрутки элемента управления `ListBox`. Их появление означает, что в окне находится больше информации, чем оно может вместить. А в правом нижнем углу диалогового окна отображается знакомый вам элемент управления, изменяющий размеры окна.

На рис. 15.19 показано то же самое диалоговое окно `UserForm`, размеры которого изменены пользователем. Обратите внимание на то, что размеры элемента управления `ListBox` были также увеличены, а кнопка Закрыть ("крестик") осталась в том же самом относительном положении. Если у вас небольшой монитор, можете сжать диалоговое окно `UserForm`.

Компакт-диск

Рассматриваемый в этом разделе пример находится на прилагаемом к книге компакт-диске в файле `resizable userform.xlsm`.

В представленном примере используется элемент управления `Label`, добавленный в диалоговое окно `UserForm` во время выполнения. Элемент управления, изменяющий размеры окна, который находится в нижнем правом углу окна, фактически представляет собой элемент управления `Label`. Он отображает единственный символ: букву "o" (символ "111") шрифта Marlett, набор символов 2. Этот элемент управления (под назва-

нием objResizer) добавляется в окно UserForm с помощью следующей процедуры UserForm_Initialize.

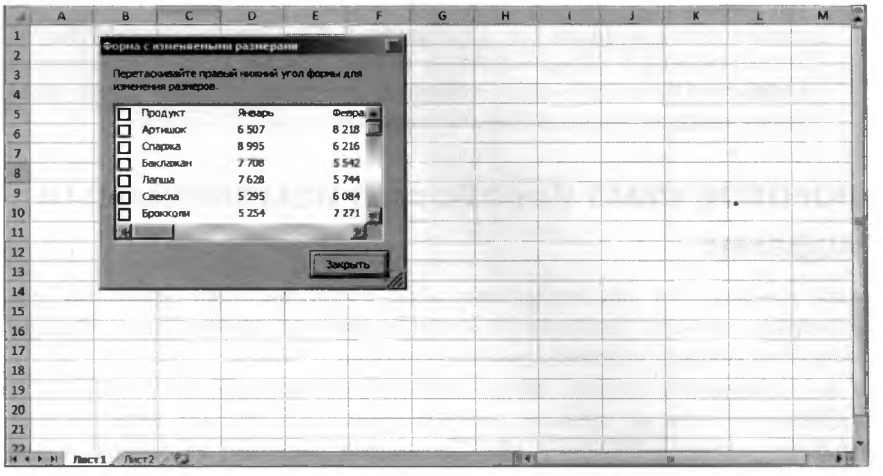


Рис. 15.18. Диалоговое окно UserForm с изменяемыми размерами

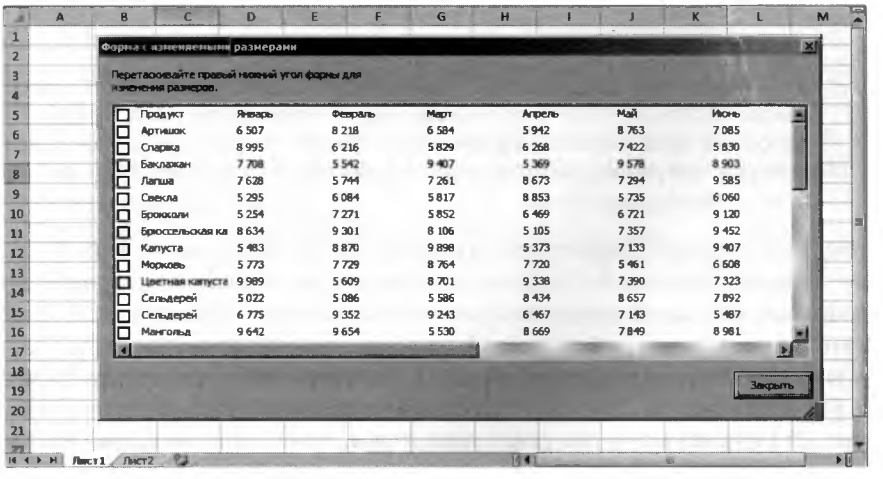


Рис. 15.19. Так выглядит диалоговое окно UserForm после увеличения размеров

```
Private Sub UserForm_Initialize()
'  Добавление элемента контроля, изменяющего размер,
'  в нижнюю правую часть окна UserForm
   Set objResizer = Me.Controls.Add("Forms.label.1", _
      MResizer, True)
   With objResizer
      .Caption = Chr(111)
      .Font.Name = "Marlett"
      .Font.Charset = 2
      .Font.Size = 14
      .BackStyle = fmBackStyleTransparent
      .AutoSize = True
      .ForeColor = RGB(100, 100, 100)
      .MousePointer = fmMousePointerSizeNWSE
```

```
      .Zorder
      .Top = Me.InsideHeight - .Height
      .Left = Me.InsideWidth - .Width
   End With
End Sub
```

Примечание

Хотя элемент управления `Label` был добавлен на этапе выполнения, код обработчика событий для объекта находится в модуле. Включение кода для несуществующего объекта не вызывает проблем.

Описанная в этом разделе техника основана на следующих фактах.

- Пользователь может перемещать элемент управления в диалоговом окне `UserForm` (см. раздел "Диалоговое окно с перемещаемыми элементами управления").
- Идентификация перемещений мыши, а также определение координат указателя мыши производится с помощью событий. В данном случае используются события `MouseDown` и `MouseMove`.
- Код VBA может изменять размер диалогового окна `UserForm` во время выполнения, но пользователю подобные возможности недоступны.

В результате творческого осмысления этих фактов можно сделать выводы о том, что можно преобразовать перемещение указателя мыши в области элемента управления `Label` в информацию, которая может использоваться для изменения размеров диалогового окна `UserForm`.

После щелчка мышью на объекте `objResizer Label` на выполнение вызывается процедура обработчика событий `objResizer_MouseDown`.

```
Private Sub objResizer_MouseDown(ByVal Button As Integer, _
   ByVal Shift As Integer, ByVal X As Single, ByVal Y As Single)
   If Button = 1 Then
      LeftResizePos = X
      TopResizePos = Y
   End If
End Sub
```

Эта процедура выполняется только в том случае, когда нажата левая кнопка мыши (при этом аргумент `Button` принимает значение `1`), а ее указатель установлен на метке `objResizer`. Координаты указателя мыши `X` и `Y`, зафиксированные во время щелчка мышью, хранятся в переменных уровня модуля: `LeftResizePos` и `TopResizePos`.

Последовательные перемещения указателя мыши приводят к вызову события `MouseMove`, после чего в игру вступает обработчик событий `objResizer_MouseMove`. Соответствующий код приводится ниже.

```
Private Sub objResizer_MouseMove(ByVal Button As Integer, _
   ByVal Shift As Integer, ByVal X As Single, ByVal Y As Single)
   If Button = 1 Then
      With objResizer
         .Move .Left + X - LeftResizePos, .Top + Y - _
            TopResizePos
         Me.Width = Me.Width + X - LeftResizePos
         Me.Height = Me.Height + Y - TopResizePos
         .Left = Me.InsideWidth - .Width
         .Top = Me.InsideHeight - .Height
      End With
End Sub
```

При внимательном изучении кода вы обнаружите, что свойства Width и Height диалогового окна UserForm были настроены на основе перемещения элемента управления objResizer Label. На рис. 15.20 показано диалоговое окно UserForm после перемещения элемента управления Label вниз и вправо.

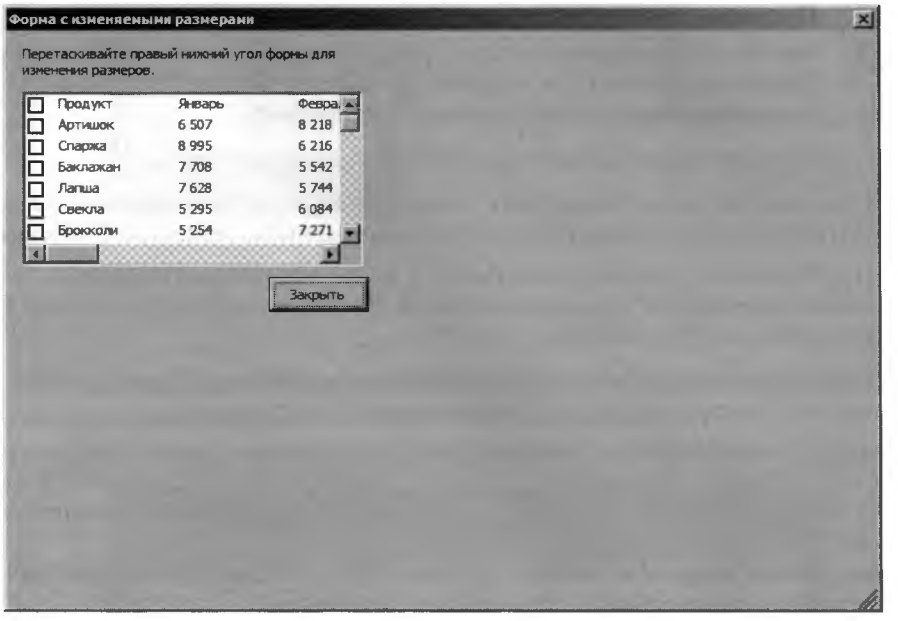


Рис. 15.20. Код VBA преобразует новые координаты элемента управления Label в значения свойств Width и Height диалогового окна UserForm

В этом случае может появиться проблема, суть которой заключается в том, что другие элементы управления в окне UserForm не соответствуют новым размерам этого окна. Элемент управления ListBox следует расширить, а элемент управления CommandButton переместить таким образом, чтобы он оставался в нижнем левом углу окна.

После изменения размеров окна UserForm настройка содержащихся в нем элементов управления производится с помощью дополнительного кода VBA. Этот код находится в процедуре обработчика событий objResizer_MouseMove. Ниже приводится код этой процедуры.

```
'   Настройка элемента управления ListBox
    On Error Resume Next
    With ListBox1
        .Width = Me.Width - 22
        .Height = Me.Height - 100
    End With
    On Error GoTo 0
'   Настройка кнопки закрытия окна
    With CloseButton
        .Left = Me.Width - 70
        .Top = Me.Height - 54
End With
```

Размеры этих двух элементов управления изменены в соответствии с размером диалогового окна UserForm (обозначается Me). После добавления нового кода диалоговое

окно приобретает "шарм и очарование". Пользователь может изменять его размеры, причем пропорционально изменяются размеры элементов управления.

Наиболее сложная задача в процессе создания диалогового окна с изменяемыми размерами — настройка элементов управления. И если их более трех, задача чрезвычайно усложняется.

Несколько кнопок с одной процедурой обработки событий

Каждый элемент управления CommandButton в диалоговом окне UserForm должен иметь собственную процедуру обработки событий. Например, если на форме находятся два элемента управления CommandButton, то необходимо создать как минимум две процедуры обработки событий.

```
Private Sub CommandButton1_Click()
'   Здесь находится код
End Sub

Private Sub CommandButton2_Click()
'   Здесь находится код
End Sub
```

Другими словами, нельзя настроить макрос так, чтобы он выполнялся после щелчка на любой кнопке CommandButton. Каждая процедура обработки события Click жестко связана с определенным элементом управления CommandButton. Однако можно заставить каждую процедуру обработки события вызывать другой макрос — при этом следует передать параметр, который будет указывать, на какой из кнопок был выполнен щелчок. В следующих примерах щелчок на одной из кнопок (CommandButton1 и CommandButton2) приведет к выполнению одной процедуры ButtonClick. Единственный аргумент сообщает процедуре ButtonClick, на какой из кнопок был сделан щелчок.

```
Private Sub CommandButton1_Click()
   Call ButtonClick(1)
End Sub

Private Sub CommandButton2_Click()
   Call ButtonClick(2)
End Sub
```

Если в диалоговом окне UserForm находится несколько элементов управления CommandButton, то создание процедур обработки событий для каждого из элементов управления может оказаться утомительным. Целесообразно использовать одну процедуру, которая определит, на какой из кнопок сделан щелчок, и в зависимости от этого выполнит соответствующие действия.

В данном разделе описывается способ, который поможет обойти это ограничение. Будет использован модуль класса с целью определения нового класса.

Компакт-диск

Рассматриваемый в этом разделе пример находится на прилагаемом к книге компакт-диске в файле multiple buttons.xlsm.

Ниже демонстрируется создание примера диалогового окна UserForm, показанного на рис. 15.21.

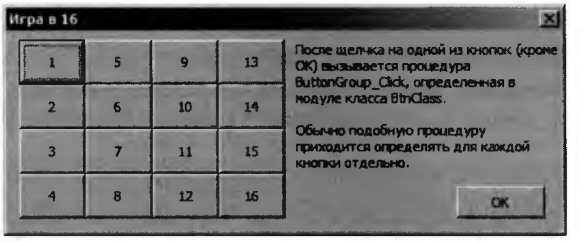


Рис. 15.21. Для нескольких кнопок используется одна процедура обработки событий

1. Создайте пользовательское диалоговое окно и добавьте в него несколько элементов управления `CommandButton` (в примере на компакт-диске содержится 16 элементов управления `CommandButton`). В рассматриваемом примере диалоговое окно называется `UserForm1`.
2. Вставьте в проект модуль класса (для этого выберите команду **Insert⇨Class Module** (Вставить⇨Модуль класса)), назовите его `BtnClass` и введите следующий код. Вам следует модифицировать процедуру `ButtonGroup_Click`.

```
Public WithEvents ButtonGroup As MsForms.CommandButton

Private Sub ButtonGroup_Click()
   Msg = "Вы щелкнули на " & ButtonGroup.Name & vbCrLf & vbCrLf
   Msg = Msg & "Название: " & ButtonGroup.Caption & vbCrLf
   Msg = Msg & "Положение слева: " & ButtonGroup.Left & vbCrLf
   Msg = Msg & "Положение сверху: " & ButtonGroup.Top
   MsgBox Msg, vbInformation, ButtonGroup.Name
End Sub
```

Совет

Описанную здесь технику можно применить для работы с другими элементами управления. Для этого следует изменить название типа в объявлении `Public WithEvents`. Например, если вы работаете с элементами управления `OptionButton` вместо `CommandButton`, используйте следующий оператор объявления:

```
Public WithEvents ButtonGroup As MsForms.OptionButton
```

3. Вставьте обычный модуль VBA и введите следующий код. Эта процедура отображает диалоговое окно `UserForm`.

```
Sub ShowDialog()
   UserForm1.Show
End Sub
```

4. В модуле кода диалогового окна `UserForm` введите приведенный ниже код `UserForm_Initialize`. Данная процедура будет запускаться при возникновении события `Initialize` для диалогового окна `UserForm`. Обратите внимание, что в коде исключается "реакция" на кнопку с названием `OKButton`. Таким образом, щелчок на кнопке `OKButton` не приведет к вызову процедуры `ButtonGroup_Click`.

```
Dim Buttons() As New BtnClass

Private Sub UserForm_Initialize()
```

```
    Dim ButtonCount As Integer
    Dim ctl As Control
'   Создание объектов Button
    ButtonCount = 0
    For Each ctl In UserForm1.Controls
       If TypeName(ctl) = "CommandButton" Then
'         Пропуск OKButton
          If ctl.Name <> "OKButton" Then
             ButtonCount = ButtonCount + 1
             ReDim Preserve Buttons(1 To ButtonCount)
             Set Buttons(ButtonCount).ButtonGroup = ctl
          End If
       End If
    Next ctl
End Sub
```

После выполнения этих инструкций можно запустить процедуру `ShowDialog`, чтобы отобразить диалоговое окно `UserForm`. Щелчок на одной из кнопок (кроме кнопки OK) приведет к выполнению процедуры `ButtonGroup_Click`. На рис. 15.22 показано окно сообщения, которое появляется после щелчка на одной из кнопок.

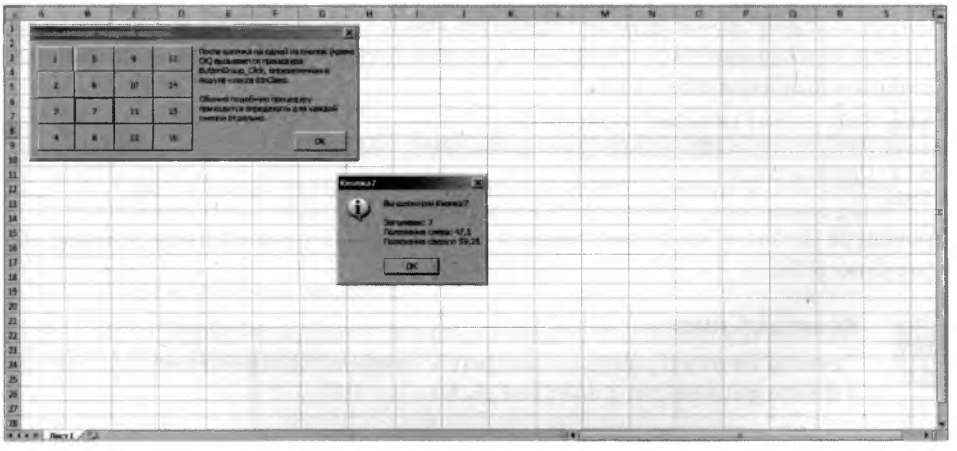


Рис. 15.22. Процедура `ButtonGroup_Click` выводит сообщение, описывающее используемую кнопку

Диалоговое окно выбора цвета

В примере этого раздела рассматривается функция, отображающая диалоговое окно (напоминает описанную ранее функцию `MyMsgBox`). В данном случае функция `GetAColor` возвращает значение цвета.

```
Public ColorValue As Variant

Function GetAColor() As Variant
  UserForm1.Show
  GetAColor = ColorValue
End Function
```

Функцию `GetAColor` можно использовать вместе с оператором, как показано ниже.

```
UserColor = GetAColor()
```

Вызов на выполнение этого оператора приводит к отображению диалогового окна UserForm. В этом окне пользователь может выбрать цвет и щелкнуть на кнопке OK. Затем функция присваивает выбранный пользователем цвет переменной UserColor. Показанное на рис. 15.23 диалоговое окно UserForm включает три элемента управления ScrollBar — по одному для каждого компонента цвета (красный, зеленый и синий). Диапазон значений для каждой полосы прокрутки изменяется от 0 до 255.

Рис. 15.23. В этом диалоговом окне можно выбрать цвет, указав красный, зеленый и синий компоненты

Модуль содержит процедуры по обработке событий ScrollBar и Change. В качестве примера ниже приводится процедура, которая вызывается в случае изменения первого элемента управления ScrollBar.

```
Private Sub ScrollBarRed_Change()
   LabelRed.BackColor = RGB(ScrollBarRed.Value, 0, 0)
   Call UpdateColor
End Sub
```

Процедура UpdateColor настраивает отображаемый образец цвета, а также обновляет значения RGB.

Компакт-диск

Пример из этого раздела находится на прилагаемом к книге компакт-диске в файле getacolor function.xlsm.

Функция GetAColor объекта UserForm имеет одну особенность: она запоминает последний выбранный цвет. По завершении выполнения функции три значения Scrollbar запоминаются в системном реестре Windows с помощью следующего кода (строка APPNAME определена в модуле Module1).

```
SaveSetting APPNAME, "Colors", "RedValue", ScrollBarRed.Value
SaveSetting APPNAME, "Colors", "BlueValue", ScrollBarBlue.Value
SaveSetting APPNAME, "Colors", "GreenValue", ScrollBarGreen.Value
```

Процедура UserForm_Initialize выбирает следующие значения, а затем присваивает их полосам прокрутки.

```
ScrollBarRed.Value = GetSetting(APPNAME, "Colors", _
   "RedValue", 128)
ScrollBarGreen.Value = GetSetting(APPNAME, "Colors", _
   "GreenValue", 128)
ScrollBarBlue.Value = GetSetting(APPNAME, "Colors", _
   "BlueValue", 128)
```

Последний аргумент для функции GetSetting представляет собой заданное по умолчанию значение, которое будет использовано, если не будет найден ключ системного ре-

стра. В этом случае значение цвета, заданного по умолчанию, равно 128 (нейтральный серый цвет).

Функции `SaveSetting` и `GetSetting` всегда используют следующий ключ системного реестра:

```
HKEY_CURRENT_USER\Software\VB and VBA Program Settings\
```

На рис. 15.24 показаны данные системного реестра, отображенные с помощью программы `Regedit.exe`.

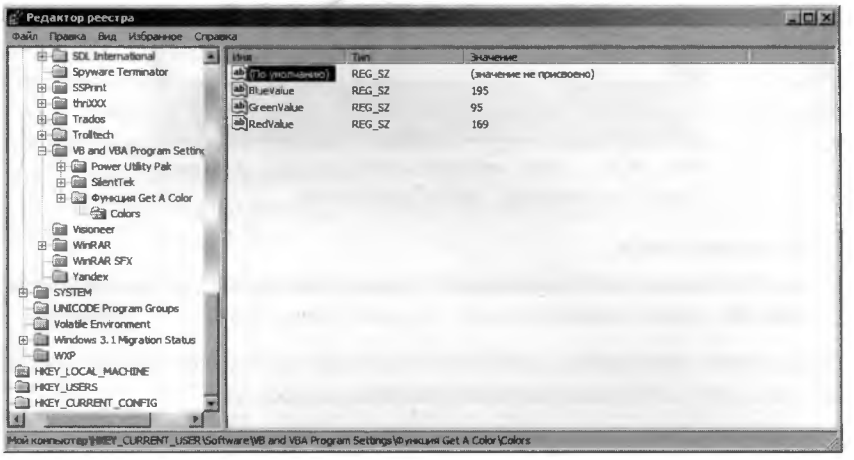


Рис. 15.24. Применяемые пользователем значения элемента управления `ScrollBar` хранятся в системном реестре и выбираются при следующем вызове функции `GetAColor`

Перекрестная ссылка

Для получения дополнительных сведений о том, как обрабатываются цвета в Excel, обратитесь к главе 30.

Отображение диаграммы в пользовательском диалоговом окне `UserForm`

Как ни странно, в Excel не предусмотрен способ отображения диаграммы в пользовательском диалоговом окне `UserForm`. Но можно скопировать диаграмму, а затем вставить ее в свойство `Picture` элемента управления `Image`. Впрочем, при этом создается статическое изображение диаграммы, которое не может учитывать произошедшие с ней изменения.

В этом разделе описывается методика, применяемая для отображения диаграмм в окне `UserForm`. На рис. 15.25 показано окно `UserForm`, отображающее диаграмму в составе объекта `Image`. При этом диаграмма фактически находится на рабочем листе, и в окне `UserForm` всегда отображается текущая диаграмма. Эта методика подразумевает копирование диаграммы во временный графический файл с последующим применением функции `LoadPicture` для указания файла, используемого свойством `Picture` элемента управления `Image`.

Рис. 15.25. Стоит немного повозиться — и в окне `UserForm` появится "живая" диаграмма

Компакт-диск

Рассматриваемый в этой главе пример находится на прилагаемом к книге компакт-диске в файле `chart in userform.xlsm`.

Для отображения диаграммы в окне `UserForm` выполните следующие действия.

1. Создайте одну или более диаграмм обычным способом.
2. Вставьте окно `UserForm` и добавьте элемент управления `Image`.
3. Напишите код VBA, сохраняющий диаграмму в виде GIF-файла, и сопоставьте GIF-файл со свойством `Picture` элемента управления `Image`.

   Для этого нужна функция VBA `LoadPicture`.
4. Добавьте необходимые элементы. Например, диалоговое окно `UserForm` в демофайле включает элементы управления, обеспечивающие изменение типа диаграммы. Можно также создать код, отображающий несколько диаграмм.

Сохранение диаграммы в виде GIF-файла

Следующий код демонстрирует создание GIF-файла (под названием `temp.gif`) на основе диаграммы (в рассматриваемом случае речь идет о первой диаграмме на листе `Данные`).

```
Set CurrentChart = Sheets("Data").ChartObjects(1).Chart
Fname = ThisWorkbook.Path & "\temp.gif"
CurrentChart.Export FileName:=Fname, FilterName:="GIF"
```

Изменение свойства `Picture` элемента управления `Image`

Пусть элемент управления `Image`, находящийся в окне `UserForm`, называется `Image1`. В этом случае следующий оператор загружает изображение (представленное переменной `Fname`) в элемент управления `Image`.

```
Image1.Picture = LoadPicture(Fname)
```

Примечание

Этот прием работает хорошо, хотя и наблюдается небольшая задержка при сохранении и последующей выборке диаграммы. В быстрой системе эта задержка незаметна.

Создание полупрозрачной формы ввода данных

Обычно пользовательская форма `UserForm` является непрозрачной и полностью скрывает все, что находится под ней. В ваших силах сделать ее полупрозрачной, в результате чего у вас появится возможность видеть находящийся под ней лист.

Для создания полупрозрачной пользовательской формы потребуется ряд функций Windows API. Уровень прозрачности может варьироваться от 0 (невидимая форма) до 255 (полностью непрозрачная форма). Значения, находящиеся в диапазоне от 0 до 255, определяют степень полупрозрачности.

На рис. 15.26 представлен пример пользовательской формы, для которой выбран уровень прозрачности, равный 128.

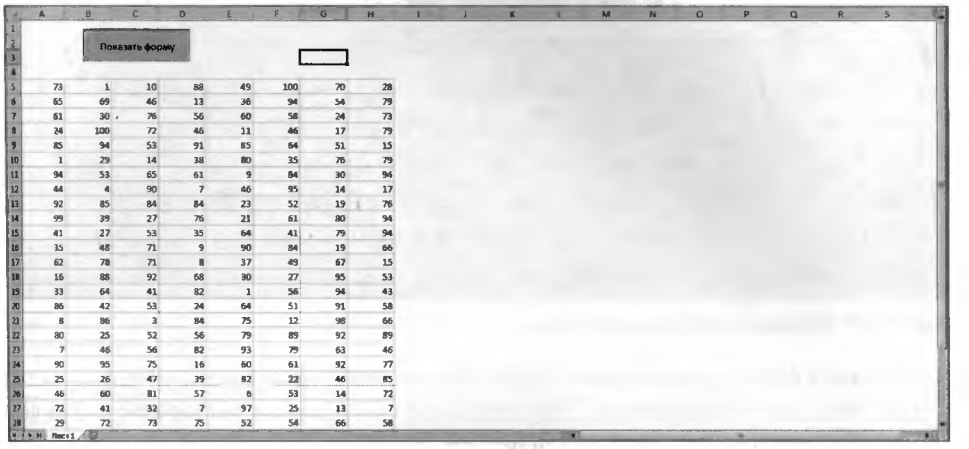


Рис. 15.26. Пример полупрозрачной формы для ввода данных

Компакт-диск

Рассматриваемый в этом разделе пример находится на прилагаемом к книге компакт-диске в файле `semitransparent userform.xlsm`.

В чем прелесть полупрозрачных форм `UserForm`? После некоторых раздумий я пришел к выводу, что свойство полупрозрачности можно использовать для создания *эффекта лайтбокса*. Вы наверняка встречали в Интернете сайты, при создании которых использовался данный эффект. Он заключается в том, что веб-страница темнеет (как будто уменьшается освещенность) в случае отображения картинки либо всплывающего окна. Этот эффект позволяет привлечь внимание пользователя к тому, что происходит на экране.

На рис. 15.27 показана книга Excel, при оформлении которой использовался эффект лайтбокса. Окно Excel темнеет, но этот эффект не влияет на окно сообщения. Как работает данный эффект? Изначально была создана пользовательская форма `UserForm` с черным фоном. Затем был разработан код, с помощью которого изменялись положение

и размеры формы UserForm таким образом, чтобы полностью перекрывалось окно Excel. Этот код приведен ниже.

```
With Me
    .Height = Application.Height
    .Width = Application.Width
    .Left = Application.Left
    .Top = Application.Top
End With
```

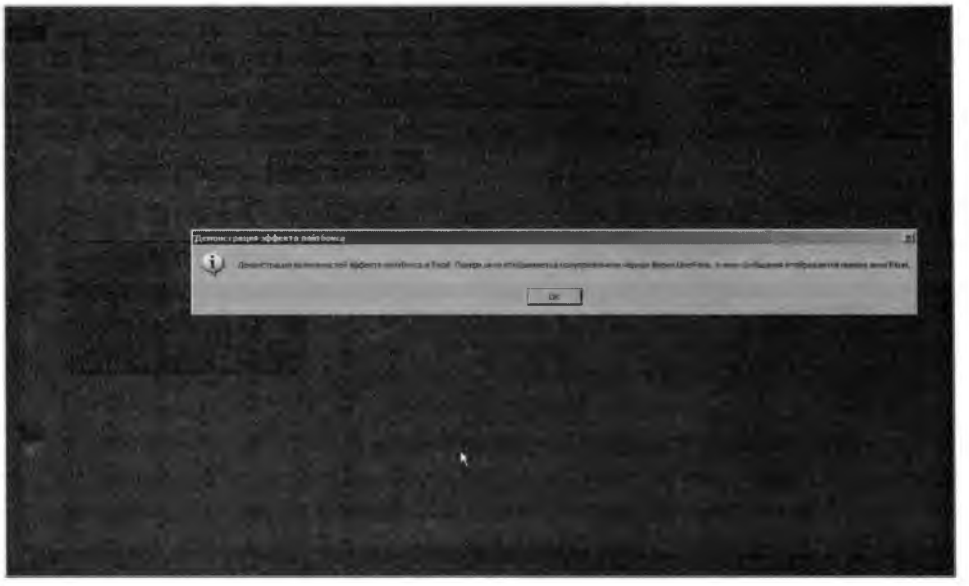


Рис. 15.27. Создание эффекта лайтбокса

После этого был придан эффект полупрозрачности форме UserForm, в результате чего окно Excel стало затененным. Окно сообщения (или другая пользовательская форма UserForm) отображается поверх полупрозрачной формы UserForm.

Компакт-диск

Рассматриваемый в этом разделе пример находится на прилагаемом к книге компакт-диске в файле excel light-box.xlsm.

Расширенная форма ввода данных

А теперь рассмотрим одно из наиболее сложных диалоговых окон UserForm, с которыми автор этой книги встречался на практике, используя их в качестве замены встроенных форм ввода данных Excel (рис. 15.28).

Подобно форме ввода данных Excel расширенная форма ввода данных поддерживает списки в рабочих листах. Как показано на рис. 15.29, эта форма ввода данных имеет особый вид, а также предлагает ряд преимуществ пользователям.

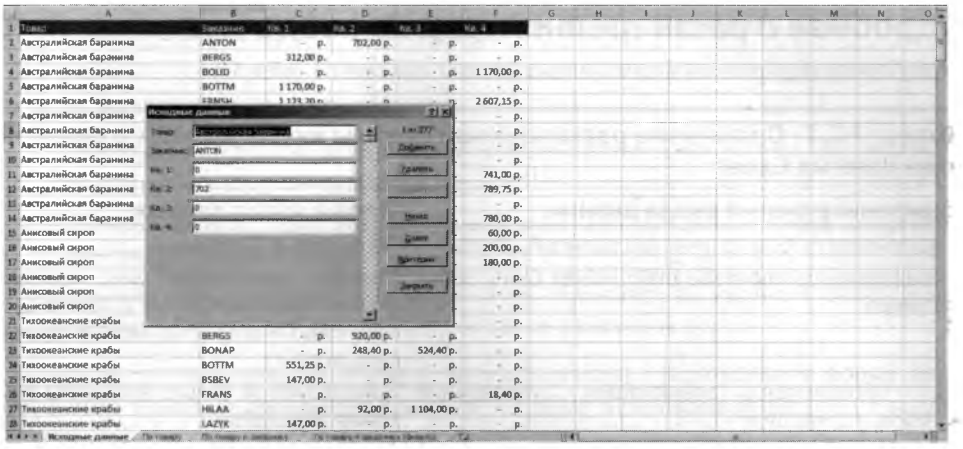


Рис. 15.28. Форма ввода данных Excel

Примечание

В Excel 2010 отобразить на экране форму ввода данных непросто. Соответствующая команда не предусмотрена интерфейсом пользователя, поэтому вам придется добавить ее на ленту или панель быстрого доступа. Для добавления команды на панель быстрого доступа щелкните на ней правой кнопкой мыши и в контекстном меню выберите пункт Настройка панели быстрого доступа (Customize Quick Access Toolbar). В отобразившемся окне Параметры Excel (Excel Options) найдите команду Форма (Form) в группе Команды не на ленте (Commands Not in the Ribbon) и добавьте ее на панель быстрого доступа.

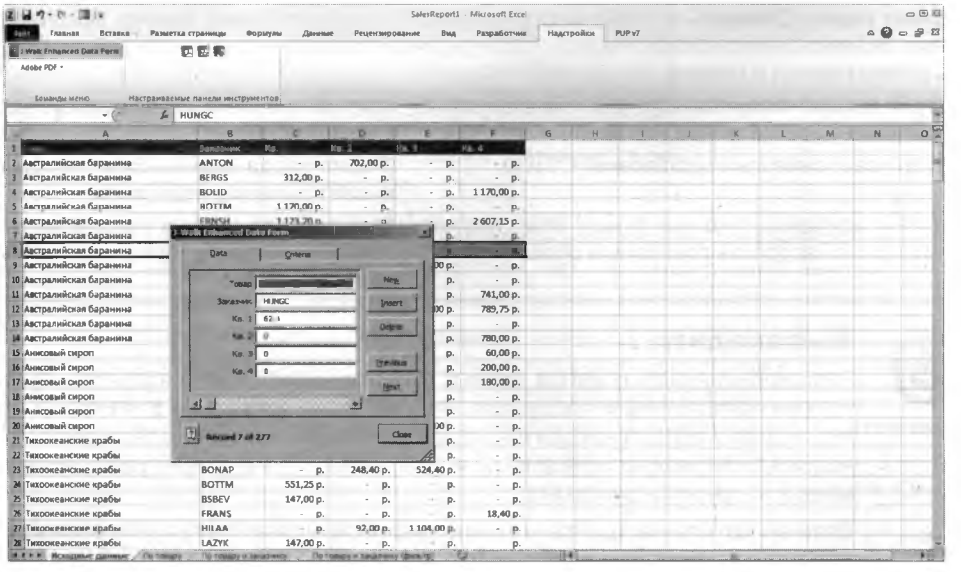


Рис. 15.29. Расширенная форма ввода данных от автора книги

Подробнее о расширенной форме ввода данных

Свойства расширенной формы ввода данных описаны в табл. 15.1.

Таблица 15.1. Сравнение расширенной формы ввода данных с формой ввода данных Excel

Расширенная форма ввода данных	Форма ввода данных
Обрабатывает любое количество записей и полей	Ограничена 32 полями
Отображаемое диалоговое окно может быть любого размера	Размер диалогового окна настраивается в зависимости от количества полей. Фактически оно может занимать весь экран!
Поля образуются элементами управления `InputBox` и `ComboBox`	Используются только элементы управления `InputBox`
Можно изменять ширину описательных заголовков столбцов	Невозможно изменять поля заголовков столбцов
Можно легко изменять используемый в диалоговых окнах язык (требуется указать пароль VBA)	Изменить язык невозможно
Запись, отображаемая в диалоговом окне, всегда присутствует на экране и при этом выделена. Благодаря этому понятно, какая запись выбрана	После выбора записей экран не прокручивается и запись не выделяется
Изначально в диалоговом окне отображается запись, находящаяся в активной ячейке	Изначально отображается первая запись в базе данных
После закрытия диалогового окна выбирается текущая запись	После закрытия выделенная область не изменяется
Позволяет вставлять новую запись в любом месте базы данных	Добавляет новые записи только в конец базы данных
Включает кнопку Отменить (Undo) для формы ввода данных, Вставить запись (Insert Record), Удалить запись (Delete Record) и Создать запись (New Record)	Включает только кнопку Восстановить (Restore)
Критерии поиска находятся на отдельной панели, благодаря чему вы всегда знаете, что ищете	Критерии поиска не всегда отображаются на экране
Поддерживает символы подстановки (*, ? и #)	Форма ввода данных Excel не поддерживает символы подстановки
Доступен исходный код на языке VBA, поэтому форму можно настроить требуемым образом	Форму ввода данных невозможно создать средствами VBA и настроить с помощью VBA

Компакт-диск

Расширенная форма ввода данных (Enhanced Data Form) является коммерческим программным продуктом. Версия этой надстройки для Excel 97 и более поздних версий находится на прилагаемом к книге компакт-диске. Этой надстройкой можно пользоваться бесплатно; также можно распространять ее среди друзей и знакомых.

Если вы хотите настроить код или диалоговое окно `UserForm`, воспользуйтесь исходным кодом на языке VBA (по весьма умеренной цене). Его можно найти на веб-сайте `http://spreadsheetpage`.

Установка надстройки Enhanced Data Form

Для работы с расширенной формой ввода данных установите надстройку Enhanced Data Form. Для этого выполните следующие действия.

1. Скопируйте в папку на жестком диске файл `dataform3.xlam`, находящийся на прилагаемом к книге компакт-диске.
2. В Excel нажмите комбинацию клавиш <Alt+TI> для отображения диалогового окна Надстройки (Add-Ins).
3. В диалоговом окне Надстройки (Add-Ins) щелкните на кнопке Обзор (Browse) и найдите файл `dataform3.xlam` в папке, указанной в п. 1.

Выполнив эти действия, вы сможете получить доступ к расширенной форме данных после выбора команды Данные⇨DataForm⇨J-Walk Enhanced DataForm. Расширенную форму данных можно использовать при работе с любым листом или таблицей.

Игра в "пятнашки"

В этом разделе вы познакомитесь с игрой в "пятнашки", реализованной в окне `UserForm` (рис. 15.30). Эта игра была придумана Нойзом Чапманом в конце XIX века. Этот пример носит не только развлекательный, но и познавательный характер.

Рис. 15.30. Играем в "пятнашки" в окне `UserForm`

Цель игры — разместить квадратики (элементы управления `CommandButton`) в порядке возрастания. После щелчка на квадратике, находящемся рядом с пустым местом, этот квадратик занимает пустое место. Элемент управления `ComboBox` (Список) позволяет сделать выбор среди трех заранее заданных конфигураций (3×3, 4×4 и 5×5). Кнопка Новая игра (New) перемешивает квадратики в произвольном порядке, а кнопка Метка (Label) позволяет отслеживать количество перемещений квадратиков.

В этом приложении используется модуль класса для обработки всех событий кнопок (см. раздел "Несколько кнопок с одной процедурой обработки событий").

Код VBA этого примера слишком длинный и поэтому здесь не приводится. Ограничимся лишь некоторыми замечаниями.

- Элементы управления `CommandButton` добавлены в окно `UserForm` с помощью кода, написанного на языке VBA. Количество и размер кнопок определяются с помощью элемента управления `ComboBox`.

- Перемешивание квадратиков осуществляется путем имитации нескольких тысяч случайных щелчков на кнопках. Еще одна возможность — простое присваивание случайных чисел, но при этом имеется вероятность появления "нерешаемых" игр.
- Пустое место на экране появляется вследствие того, что свойству `Visible` элемента управления `CommandButton` присваивается значение `False`.
- Модуль класса включает одну процедуру обработки событий (`MouseUp`), которая вызывается после щелчка мышью на кнопке.
- После того как пользователь щелкнет на кнопке `CommandButton`, происходит замена ее заголовка (`Caption`) скрытой кнопкой. Сами кнопки при этом не перемещаются.

Компакт-диск

Рассматриваемый в этом разделе пример находится на прилагаемом к книге компакт-диске в файле `sliding tile puzzle.xlsm`.

Играем в видеопокер в окне `UserForm`

И напоследок протестируем "развлекательные" возможности Excel. На рис. 15.31 показана форма `UserForm` в стиле видеопокера — игры из казино. Игра обладает следующими свойствами:

- возможность выбора среди двух игр — "Joker's Wild" либо "Jacks Or Better";
- диаграмма, отображающая историю ваших побед (или поражений);
- возможность изменения выплат;
- справка (отображается на листе);
- кнопка тревоги, которая позволяет быстро скрыть форму `UserForm`.

Всего этого вы не найдете в настоящем казино.

Компакт-диск

Рассматриваемый в этом разделе пример можно найти на прилагаемом к книге компакт-диске в файле `video poker.xlsm`.

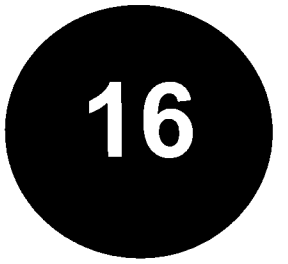
Рис. 15.31. Модернизированная игра в видеопокер

Как и следовало ожидать, код примера оказался слишком большим для публикации в книге. Обратитесь к прилагаемому к книге компакт-диску — и узнаете много полезного.

Часть V

Профессиональные методы программирования

В этой части...



Разработка утилит Excel с помощью VBA

В этой главе...

- Об утилитах Excel
- Создание утилит с помощью VBA
- Признаки хорошей утилиты
- Утилита Text Tools
- Дополнительно об утилитах Excel

В этой главе вы познакомитесь с методами разработки утилит Excel.

Об утилитах Excel

Основное назначение *утилиты* — расширение возможностей программы, обеспечение и упрощение доступа к имеющимся функциям программы. Утилита не является конечным продуктом (как, например, приложение квартального отчета). Это инструмент, который позволяет создавать конечный продукт (например, квартальный отчет). Утилита Excel (почти всегда) — это надстройка, которая расширяет возможности программы Excel, добавляя в нее новые функциональные средства.

Excel — великий продукт, но нет предела совершенству. Многие пользователи продолжают добавлять в программу различные свойства. Например, по мнению некоторых пользователей, часто занимающихся вводом дат, будет полезным всплывающий календарь, автоматизирующий ввод данных в ячейки. А другим пользователям хотелось бы иметь простой способ экспорта диапазона данных в отдельный файл. Все это примеры средств, которые в настоящее время в Excel недоступны, но могут быть добавлены с помощью соответствующей утилиты.

С другой стороны, утилиты не должны быть слишком сложными. Многие из популярных утилит очень просты. Например, как вы уже, наверное, заметили, в Excel 2010 отсутствует команда ленты, с помощью которой отображается (либо скрывается) размет-

ка листа. Если вам не нравятся пунктирные линии разметки, откройте диалоговое окно Параметры Excel и воспользуйтесь соответствующей опцией для отключения разметки. К сожалению, команду отображения и сокрытия разметки листа невозможно добавить на ленту либо на панель быстрого доступа.

Ниже представлен пример простой процедуры VBA, которая отображает и скрывает разметку страницы в активном окне.

```
Sub TogglePageBreaks()
   With ActiveSheet
      .DisplayPageBreaks = Not .DisplayPageBreaks
   End With
End Sub
```

Этот макрос можно сохранить в персональной книге макросов, чтобы всегда иметь к нему доступ. Вы получите быстрый доступ к макросу, если назначите ему значок на панели инструментов и определите команду в меню, опцию в контекстном меню или даже измените ленту.

Как видите, создание утилит Excel — это превосходный способ сделать хороший продукт совершенным.

Создание утилит с помощью VBA

В 1992 году увидела свет Excel 5 — первая версия процессора электронных таблиц от Microsoft, в которой появилась поддержка VBA. При изучении бета-версии Excel 5 многих поразил потенциал VBA, который на световые годы опережал встроенный язык Excel для создания макросов (XLM) и зачислил Excel в ряды лидеров среди программ управления электронными таблицами, особенно в вопросах программирования новых возможностей.

В процессе изучения VBA я создал коллекцию утилит Excel, которые были написаны исключительно с использованием VBA. Я понял, что могу изучить язык программирования быстрее, если поставлю перед собой реальную цель. Результатом стало создание пакета Power Utility Pak для Excel.

При изучении языка VBA следует учитывать такие моменты.

- VBA может оказаться трудным в изучении, но со временем его использование упрощается.
- Экспериментирование является основным способом изучения VBA. Проект обычно вырастает из десятка экспериментов по созданию кода, что в итоге приводит к появлению завершенного продукта.
- VBA предоставляет возможность расширять возможности Excel, сохраняя внешний вид диалоговых окон и панелей инструментов.
- Excel может выполнять любые задачи. Если разработка зашла в тупик, то существует вероятность, что к решению ведет другой путь.

Далеко не все продукты располагают таким богатым набором инструментов, который позволяет конечному пользователю значительно расширить возможности программы.

Признаки хорошей утилиты

Утилита Excel должна упрощать выполнение задачи, причем делать это более эффективно, чем смог бы сделать вручную пользователь. Если утилита разрабатывается для

конечных пользователей, то что же делает ее ценной? Ниже приведены характеристики хорошей утилиты.

- **Утилита привносит что-либо новое в Excel.** Это может быть и новая возможность, и комбинация существующих функций, и более простой способ использования одного из существующих средств.
- **Утилита имеет универсальную природу.** В идеальном случае она должна обладать возможностями, которые работают в различных средах. Конечно, создать утилиту общего назначения намного сложнее, чем создать утилиту, работающую в строго определенной среде.
- **Утилита обладает гибкостью.** Наилучшие утилиты предоставляют большое количество параметров, что позволяет использовать их для обработки широкого диапазона ситуаций.
- **Утилита выглядит и работает, как встроенная команда Excel.** Утилиты должны выглядеть и вести себя так же, как и встроенные команды Excel, хотя можно добавлять и индивидуальные настройки.
- **Утилита предоставляет пользователям необходимую справочную информацию.** Другими словами, следует создать документацию, к которой пользователь сможет обратиться в случае необходимости.
- **Утилита отслеживает появление ошибок.** Конечный пользователь не должен видеть сообщений об ошибках VBA. Любое сообщение об ошибке, которое увидит пользователь, должно создаваться автором утилиты.
- **Утилита предоставляет способ отмены результата собственных действий.** Пользователи, которых не удовлетворяет результат работы утилиты, должны иметь возможность отменить внесенные изменения.

Утилита Text Tools

В этом разделе рассматривается утилита Excel, которая была разработана в качестве составной части пакета Power Utility Pak. Утилита Text Tools (Текстовые инструменты) позволяет пользователю манипулировать текстом в выделенном диапазоне ячеек. В частности, эта утилита поддерживает такие операции:

- изменение регистра текста (верхний регистр, нижний регистр или правильный регистр);
- добавление символов в начале, в конце или в указанной позиции;
- удаление текста в начале, в конце или в указанной позиции строки;
- удаление лишних пробелов (или всех пробелов);
- удаление символов из текста (не выводимых на печать символов, символов алфавита, нечисловых символов, не алфавитных символов либо числовых символов).

На рис. 16.1 показана утилита Text Tools в действии.

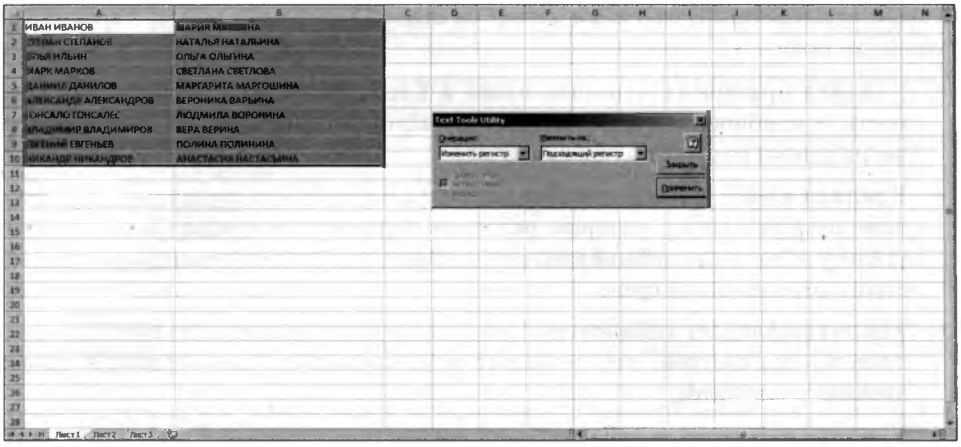


Рис. 16.1. Утилита Text Tools применяется для изменения регистра символов выделенного текста

Компакт-диск

Утилиту Text Tools можно найти на прилагаемом к книге компакт-диске. Она представляет собой отдельный инструмент, который также входит в состав пакета Power Utility Pak. Файл этой утилиты под названием `text tools.xlam` является стандартной надстройкой Excel. После ее инсталляции на ленте появляется новая команда: Главная⇨Text Tools Utilities⇨Text Tools. Проект VBA не защищен паролем, поэтому можно свободно просматривать код, что позволяет понять, как он работает.

Обоснование

Excel содержит ряд текстовых функций, позволяющих манипулировать текстовыми строками несколькими способами. Например, можно преобразовывать символы в ячейке в верхний регистр (ПРОПИСН), добавлять символы в текст (СЦЕПИТЬ), удалять из текстовой строки пробелы (СЖПРОБЕЛЫ) и т.д. Но для того чтобы выполнить одну из перечисленных операций, необходимо создать формулы, скопировать их, преобразовать в значения и вставить эти значения вместо первоначального текста. Другими словами, Excel не предоставляет простого способа управления текстом. Было бы неплохо, чтобы программа имела инструменты работы с текстом, которые не требуют использования формул.

Кстати, многие прогрессивные идеи начинаются со слов "Если бы...".

Назначение проекта

Первый этап в разработке утилиты — точно определить ее назначение. Ниже приведен первоначальный план, представленный в виде двенадцати целей.

- Основные свойства утилит были описаны в начале этого раздела.
- Пользователь имеет возможность выполнять изменения по отношению как к текстовым, так и к нетекстовым ячейкам.
- Утилита должна выглядеть и вести себя так же, как и остальные команды Excel. Другими словами, утилита будет отображать диалоговое окно, которое выглядит так же, как и другие диалоговые окна Excel.

- Она может быть реализована в виде подключаемого модуля и быть доступной с ленты.
- Утилита может оперировать текущими выделенными ячейками (включая несколько выделенных областей) и позволяет изменять выделенный диапазон в то время, когда отображается диалоговое окно.
- Утилита запоминает последнюю операцию, а также отображает настройки при следующем вызове диалогового окна.
- Утилита не должна влиять на ячейки, содержащие формулы.
- Она должна быть быстрой и эффективной. Например, если пользователь выделяет весь столбец, утилита игнорирует пустые ячейки в этом столбце.
- Утилита должна применять немодальные диалоговые окна, которые всегда отображаются на экране и готовы к применению.
- Она должна быть компактной и не перекрывать данные рабочего листа.
- Утилита должна разрешать пользователям отменять внесенные ранее изменения.
- Пользователи утилиты должны иметь доступ к исчерпывающей справочной информации.

Рабочая книга утилиты

Утилита Text Tools реализована в виде файла надстройки XLAM. В процессе разработки я использовал рабочую книгу с поддержкой макросов XLSM. Как только я убедился в том, что все работает, я сохранил рабочую книгу в виде надстройки.

Рабочая книга утилиты Text Tools состоит из следующих компонентов.

- **Один лист.** Каждая рабочая книга должна иметь как минимум один лист. В данном случае этот лист используется для хранения процедур, выполняющих отмену совершенных ранее действий.
- **Один модуль кода VBA.** Этот модуль содержит объявления общедоступных переменных и констант. Кроме того, в нем находится код отображения пользовательской формы и управления процедурой отмены действия. Весь код, выполняющий конечные действия, сохранен в модуле пользовательской формы.
- **Одно диалоговое окно `UserForm`.** Содержит пользовательскую форму. Код, выполняющий обработку текста, находится в модуле кода для `UserForm`.

Примечание

Этот файл также включает результат некоторых изменений, выполненных вручную. Цель этих изменений — отображение команд на ленте (см. также раздел "Добавление кода RibbonX"). К сожалению, невозможно изменить ленту Excel с помощью VBA.

Установка надстроек

Для установки любой надстройки, в том числе надстройки `text tools.xlam`, выполните следующие действия.

1. Выберите команду Файл⇨Параметры Excel (File⇨Excel Options).
2. В окне Параметры Excel (Excel Options) выберите вкладку Надстройки (Add-ins).

3. В раскрывающемся списке Управление (Manage) выберите пункт Надстройки Excel (Excel Add-Ins) и щелкните на кнопке Перейти (Go to), после чего отобразится диалоговое окно Надстройки (Add-Ins).
4. Если устанавливаемая надстройка находится в списке Доступные надстройки (Add-Ins Available), установите рядом с ней флажок. Если же надстройка отсутствует в этом списке, щелкните на кнопке Просмотр (Browse) для поиска соответствующего файла `XLAM` или `XLA`.
5. Щелкните на кнопке OK для установки надстройки.

   Надстройка будет установленной до тех пор, пока вы не удалите ее из списка.

Можете пропустить пп. 1–3 приведенной выше инструкции, нажав комбинацию клавиш <Alt+TI> для открытия окна Надстройки. Эта комбинация работает во всех версиях Excel.

Как работает утилита

Надстройка Text Tools включает код RibbonX, который создает новый элемент на ленте: Главная⇨Utilities⇨Text Tools. Выбор этого элемента приводит к выполнению процедуры `StartTextTools`, которая вызывает процедуру `ShowTextToolsDialog`.

Перекрестная ссылка

Наличие двух вспомогательных процедур для этой утилиты (`StartTextTools` и `ShowTextToolsDialog`) обосновывается в разделе "Добавление кода RibbonX".

Пользователь может выбрать необходимые опции для изменения текста и щелкнуть на кнопке Применить (Apply), чтобы применить их. Изменения сразу же будут представлены в рабочей книге. Диалоговое окно продолжает отображаться на экране. Каждую операцию можно отменить. Пользователь также имеет возможность выполнить некоторые дополнительные действия. Щелчок на кнопке Справка приведет к отображению диалогового окна справочного руководства. Щелчок на кнопке Закрыть вызывает закрытие диалогового окна. А поскольку вы имеете дело с немодальным диалоговым окном, у вас есть возможность продолжать работать в Excel в то время как будет отображаться диалоговое окно. В этом смысле оно ведет себя подобно панели инструментов.

Пользовательская форма утилиты

Создание утилиты обычно начинается с разработки пользовательского интерфейса, который в данном случае определяется основным диалоговым окном. Операция по созданию диалогового окна заставляет повторно пересмотреть все концепции проекта.

На рис. 16.2 показано диалоговое окно `UserForm` утилиты Text Tools.

Нетрудно заметить, что расположение отдельных элементов управления в окне `UserForm` различно в режиме проектирования и выполнения утилиты. Это происходит потому, что их расположение динамически изменяется в коде. Ниже описаны главные элементы диалогового окна.

- **Список Операция (Operation).** Этот список находится в левой части формы и предназначен для указания типа выполняемой операции.
- **Список Процедура1.** Точно указывает текстовую операцию, соответствующую указанному в предыдущем списке типу.

Рис. 16.2. Диалоговое окно `UserForm` для утилиты Text Tools

- **Список Процедура2.** Только две текстовые операции нуждаются в этом списке. С его помощью указываются дополнительные инструкции для текстовой команды.
- **Флажок.** Позволяет игнорировать нетекстовые ячейки.
- **Кнопка Справка (Help).** Это элемент управления `CommandButton`, который содержит изображение. Щелчок на указанной кнопке приводит к отображению справочной системы утилиты.
- **Кнопка Закрыть (Close).** Щелчок на данной кнопке приводит к выгрузке пользовательской формы.
- **Кнопка Применить (Apply).** Щелчок на этом элементе управления `CommandButton` приводит к применению текстовых параметров, которые установлены в пользовательской форме.
- **Индикатор выполнения команды.** Состоит из элементов управления `Label` и `Frame`.
- **Текстовое поле.** Используется при добавлении текста.

На рис. 16.3 показана пользовательская форма при выборе самых разных текстовых инструментов. Обратите внимание на изменение параметров в окне при выборе другой команды.

Модуль `Module1`

Модуль `Module1` объявляет данные, содержит простую процедуру, которая завершает утилиту, а также включает процедуру, выполняющую операции отмены.

Объявление данных

Ниже приведены объявления, которые находятся в верхней части модуля `Module1`.

```
Public Const APPNAME As String = "Text Tools"
Public Const PROGRESSTHRESHOLD = 2000
Public UserChoices(1 To 8) As Variant ' хранит последние варианты,
                                      ' выбранные пользователем
Public UndoRange As Range      ' для выполнения отмены
Public UserSelection As Range ' для выполнения отмены
```

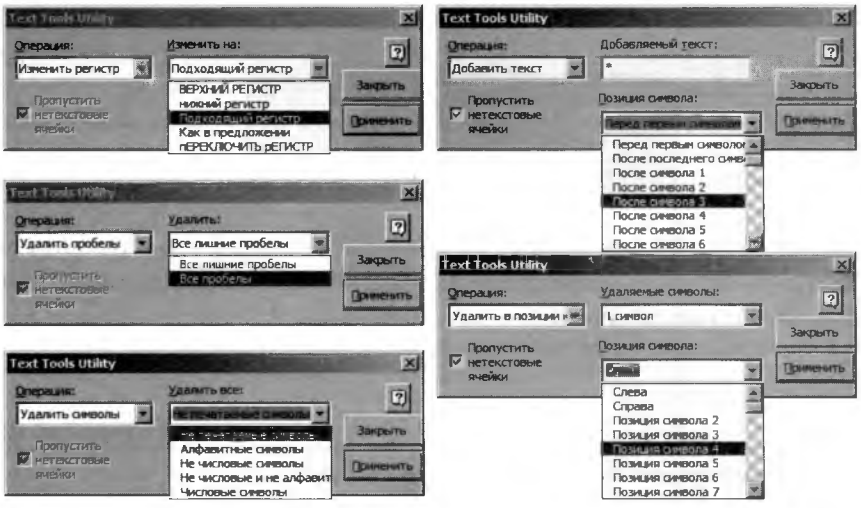


Рис. 16.3. Содержимое окна зависит от выбранных пользователем параметров

Вначале объявляется глобальная константа, которая содержит строку, хранящую имя приложения. Эта строка используется в окнах сообщений и в качестве заголовка создаваемой команды меню.

Константа `PROGRESSTHRESHHOLD` указывает количество ячеек, для которых отображается индикатор выполнения операции. В нашем случае индикатор отображается только при обработке более 2000 ячеек.

Массив `UserChoices` содержит значения каждого элемента управления. Эта информация сохраняется в системном реестре и запрашивается при выполнении утилиты.

Кроме того, здесь представлены два объекта `Range`. Их назначение — хранить информацию, необходимую для отмены выполненной операции.

Процедура `ShowTextToolsDialog`

Ниже приведен код процедуры `ShowTextToolsDialog`.

```
Sub ShowTextToolsDialog()
   Dim InvalidContext As Boolean
   If Val(Application.Version) < 12 Then
      MsgBox "Утилита требует Excel версии 2007 или более поздней.", _
         vbCritical
      Exit Sub
   End If
   If ActiveSheet Is Nothing Then InvalidContext = True
   If TypeName(ActiveSheet) <> "Worksheet" Then _
      InvalidContext = True
   If InvalidContext Then
      MsgBox "Выберите ячейки в диапазоне.", vbCritical, APPNAME
   Else
      UserForm1.Show vbModeless
   End If
End Sub
```

Как видите, процедура очень проста. Она начинает работу с проверки версии Excel. Если используемая версия более ранняя, чем Excel 2007, отображается сообщение о том, что требуется Excel 2007 или более поздняя версия.

Примечание

Конечно, можно было создать утилиту, которая работает с предыдущими версиями, но я решил не усложнять задачу, а ограничиться версией Excel 2007.

Если вы работаете с Excel 2007, процедура `ShowTextToolsDialog` проверяет, активен ли лист, а затем еще раз проверят его принадлежность к рабочим листам. Если хотя бы одно условие не выполнено, то переменной `InvalidContext` присваивается значение `True`. Конструкция `If-Then-Else` проверяет значение этой переменной и отображает сообщение (рис. 16.4) или пользовательскую форму. Обратите внимание, что метод `Show` использует аргумент `0`, что указывает на *немодальность* формы (благодаря этому пользователь может работать в Excel и в диалоговом окне одновременно).

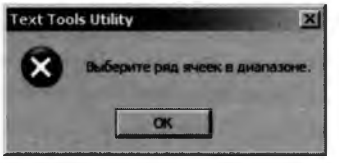


Рис. 16.4. Это сообщение отображается в случае, когда данная рабочая книга не является активной (либо не является активным текущий рабочий лист)

Обратите внимание, что в коде не проверяется выделение диапазона ячеек. Соответствующая процедура обработки ошибок вызывается после щелчка мышью на кнопке Применить (Apply).

Совет

Для вызова этой процедуры (в целях тестирования) была назначена комбинация клавиш <Ctrl+Shift+T>. Это связано с тем, что модификацию ленты я отложил напоследок, а тестировать утилиту как-то нужно. После добавления на ленту соответствующей кнопки я отказался от этой комбинации клавиш.

Чтобы назначить комбинацию клавиш макросу, следует нажать <Alt+F8> для отображения диалогового окна Макрос (Macro). В поле Имя макроса (Macro Name) введите название `ShowTextToolsDialog` и щелкните на кнопке Параметры (Options). В окне Параметры макроса (Macro Options) назначьте указанную выше комбинацию клавиш.

Процедура `UndoTextTools`

Процедура `UndoTextTools` вызывается после щелчка на кнопке Отмена (Undo) либо нажатия клавиш <Ctrl+Z>. Работа этой процедуры описывается в разделе "Отмена ранее выполненных действий".

Модуль кода `UserForm1`

Все важные операции выполняются с помощью кода, сохраненного в модуле `UserForm1`. Ниже вкратце описаны все процедуры этого модуля. Сам код модуля слишком длинный, чтобы приводить его, но вы всегда можете открыть файл `text tools.xlam` на прилагаемом к книге компакт-диске и ознакомиться с ним.

Процедура `UserForm_Initialize`

Эта процедура выполняется перед отображением пользовательской формы. Она изменяет размер формы, чтобы поместить в нее все необходимые параметры (их значения сохраняются в системном реестре). Данная процедура также определяет список пунктов меню (под названием `ComboBoxOperation`), которые указывают выполняемую операцию. Эти пункты меню перечислены ниже:

- изменение регистра;
- добавление текста;
- удаление на основе указанной позиции;
- удаление пробелов;
- удаление символов.

Процедура `ComboBoxOperation_Change`

Эта процедура выполняется после выбора пользователем операции в раскрывающемся списке. Она отображает или скрывает элементы управления. Например, если пользователь выбрал опцию **Изменение регистра**, отображается первый список (элемент управления `ComboBox`), в котором можно выбрать один из следующих элементов:

- `ВЕРХНИЙ РЕГИСТР`;
- `нижний регистр`;
- `Подходящий регистр`;
- `Регистр предложения`;
- `пЕРЕКЛЮЧИТЬ рЕГИСТР`.

Процедура `ApplyButton_Click`

Эта процедура вызывается после щелчка мышью на кнопке **Применить** (Apply). Она выполняет проверку ошибок, позволяющую гарантировать выбор корректного диапазона, и вызывает функцию `CreateWorkRange`, которая позволяет убедиться в том, что не обрабатываются пустые ячейки. (См. также раздел "Повышение эффективности утилиты".)

Процедура `ApplyButton_Click` также вызывает процедуру `SaveForUndo`, которая сохраняет текущие данные, требуемые для выполнения операции отмены. (См. также раздел "Отмена ранее выполненных действий".)

Затем эта процедура использует конструкцию `Select Case` для вызова соответствующей процедуры, которая выполняет нужную операцию — вызывает одну из следующих процедур:

- `ChangeCase`;
- `AddText`;
- `RemoveText`;
- `RemoveSpaces`;
- `RemoveCharacters`.

Некоторые из этих процедур вызывают функции. Например, процедура `ChangeCase` может вызывать функцию `ToggleCase` либо `SentenceCase`.

Процедура `CloseButton_Click`

Выполняется после щелчка на кнопке Закрыть (Close). Она сохраняет параметры текущего состояния в системном реестре и выгружает пользовательскую форму.

Процедура `HelpButton_Click`

Выполняется после щелчка на кнопке Справка. Отображает содержимое справочного файла (стандартный компилированный HTML-файл справки).

Повышение эффективности утилиты

Процедуры утилиты выполняют возложенные на них задачи, циклически просматривая диапазон ячеек. При этом нет никакого смысла просматривать неизменяющиеся ячейки, например пустые ячейки или ячейки с формулами.

Как уже отмечалось, процедура `ApplyButton_Click` вызывает процедуру-функцию `CreateWorkRange`. Эта функция создает и возвращает объект `Range`, в котором нет пустых ячеек и ячеек с формулами. Например, пусть столбец A содержит текст в диапазоне A1:A12. Если пользователь выделит весь столбец, то функция `CreateWorkRange` преобразует диапазон всего столбца в диапазон без пустых ячеек (т.е. A1:A12). В результате приложение будет выполняться эффективнее, не обрабатывая ненужных данных (пустые ячейки и формулы).

Функция `CreateWorkRange` использует два аргумента.

- `Rng`. Объект `Range`, представляющий выделенный пользователем диапазон данных.
- `TextOnly`. Булево значение. Если оно равно `True`, функция возвращает ячейки только с текстовыми значениями. В противном случае возвращаются все непустые ячейки.

```
Private Function CreateWorkRange(Rng, TextOnly)
'  Создает и возвращает объект Range
   Set CreateWorkRange = Nothing
'  Единственная ячейка, включающая формулу
   If Rng.Count = 1 And Rng.HasFormula Then
      Set CreateWorkRange = Nothing
      Exit Function
   End If
'  Единственная ячейка либо одна объединенная ячейка
   If Rng.Count = 1 Or Rng.MergeCells = True Then
      If TextOnly Then
         If Not IsNumeric(Rng(1).Value) Then
            Set CreateWorkRange = Rng
            Exit Function
         Else
            Set CreateWorkRange = Nothing
            Exit Function
         End If
      Else
         If Not IsEmpty(Rng(1)) Then
            Set CreateWorkRange = Rng
            Exit Function
         End If
      End If
   End If
   On Error Resume Next
   Set Rng = Intersect(Rng, Rng.Parent.UsedRange)
   If TextOnly = True Then
```

```
            Set CreateWorkRange = Rng.SpecialCells(xlConstants, _
               xlTextValues)
            If Err <> 0 Then
               Set CreateWorkRange = Nothing
               On Error GoTo 0
               Exit Function
            End If
            Else
               Set CreateWorkRange = Rng.SpecialCells _
                  (xlConstants, xlTextValues + xlNumbers)
            If Err <> 0 Then
               Set CreateWorkRange = Nothing
               On Error GoTo 0
               Exit Function
            End If
         End If
      End Function
```

Примечание

Функция `CreateWorkRange` практически повсеместно использует свойство `SpecialCells`. Для получения дополнительных сведений о свойстве `SpecialCells` создайте макрос, который выполняет различные выделения в диалоговом окне Excel Выделение группы ячеек (Go To Special). Для перехода в это окно нажмите клавишу <F5> и щелкните на кнопке Выделить (Special) в диалоговом окне Переход (Go To).

Диалоговое окно **Выделение группы ячеек** имеет одну особенность. Обычно оно работает с текущим выделенным диапазоном. Например, если выделен целый столбец, то результатом будет подмножество ячеек этого столбца. Но если выделена одна ячейка, то диалоговое окно работает со всем листом. Именно поэтому функция `CreateWorkRange` проверяет количество ячеек, которые составляют диапазон, переданный функции в качестве аргумента.

Сохранение настроек утилиты

Утилита Text Tools характеризуется одним очень интересным свойством: она сохраняет последние используемые настройки. Это невероятно удобно, поскольку чаще всего требуется повторно выполнять одни и те же действия.

Последние настройки приложения сохраняются в системном реестре. После щелчка на кнопке **Закрыть** выполняется функция `SaveSettings`, которая отвечает за сохранение значений каждого элемента управления. При запуске утилиты выполняется функция `GetSettings`, которая запрашивает настройки из системного реестра.

В системном реестре настройки утилиты сохраняются в следующем разделе.

```
HKEY_CURRENT_USER\Software\VB and VBA Program Settings\ _
   Text Tools Utility\Settings
```

На рис. 16.5 показаны эти настройки (в окне редактора системного реестра — `regedit.exe`).

В процессе просмотра кода утилиты Text Tools вы обнаружите, что для сохранения настроек используется восьмиэлементный массив (под именем `UserChoices`). В принципе, для хранения настроек можно воспользоваться переменными, но использование массива немного облегчает работу программиста.

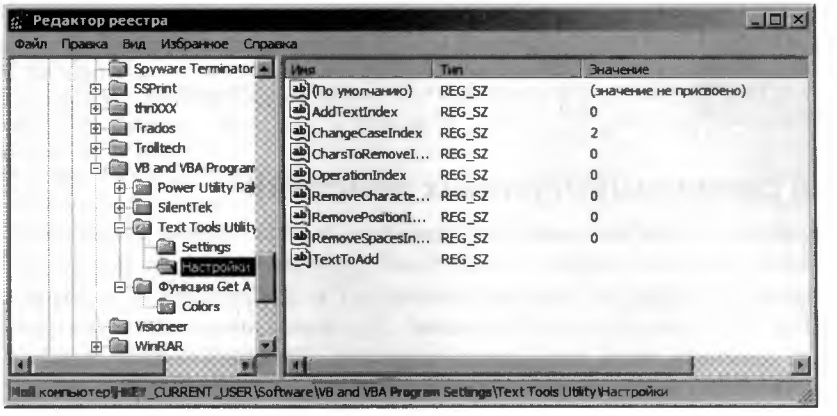


Рис. 16.5. Программа редактора системного реестра применяется для просмотра сохраненных в реестре настроек

Следующий код VBA считывает настройки из системного реестра и сохраняет их в массиве `UserChoices`.

```
' Получение предыдущих настроек
  UserChoices(1) = GetSetting(APPNAME, "Настройки", _
     "OperationIndex", 0)
  UserChoices(2) = GetSetting(APPNAME, "Настройки", _
     "ChangeCaseIndex", 0)
  UserChoices(3) = GetSetting(APPNAME, "Настройки", _
     "TextToAdd", "")
  UserChoices(4) = GetSetting(APPNAME, "Настройки", _
     "AddTextIndex", 0)
  UserChoices(5) = GetSetting(APPNAME, "Настройки", _
     "CharsToRemoveIndex", 0)
  UserChoices(6) = GetSetting(APPNAME, "Настройки", _
     "RemovePositionIndex", 0)
  UserChoices(7) = GetSetting(APPNAME, "Настройки", _
     "RemoveSpacesIndex", 0)
  UserChoices(8) = GetSetting(APPNAME, "Настройки", _
     "RemoveCharactersIndex", 0)
  cbSkipNonText.Value = GetSetting(APPNAME, "cbSkipNonText", 0)
```

Следующий код вызывается после того, как диалоговое окно будет закрыто. Образующие этот код операторы выбирают значения из массива `UserChoices`, сохраняя их в системном реестре.

```
' Сохранение настроек
  SaveSetting APPNAME, "Настройки", "OperationIndex", _
     UserChoices(1)
  SaveSetting APPNAME, "Настройки", "ChangeCaseIndex", _
     UserChoices(2)
  SaveSetting APPNAME, "Настройки", "TextToAdd", _
     UserChoices(3)
  SaveSetting APPNAME, "Настройки", "AddTextIndex", _
     UserChoices(4)
  SaveSetting APPNAME, "Настройки", "CharsToRemoveIndex", _
     UserChoices(5)
  SaveSetting APPNAME, "Настройки", "RemovePositionIndex", _
     UserChoices(6)
  SaveSetting APPNAME, "Настройки", "RemoveSpacesIndex", _
```

```
    UserChoices(7)
SaveSetting APPNAME, "Настройки", "RemoveCharactersIndex", _
    UserChoices(8)
SaveSetting APPNAME, "Настройки", "cbSkipNonText", _
    cbSkipNonText.Value * -1
```

Отмена ранее выполненных действий

К сожалению, в Excel не существует прямого способа отмены операции средствами VBA. Конечно, написать макрос с подобными функциями можно, но это не так уж и просто. Также, в отличие от свойства отмены в Excel, операция отмены, реализуемая утилитой Text Tools, является одноуровневой. Другими словами, пользователь этой утилиты может отменить лишь последнюю операцию. Дополнительные сведения относительно применения свойства отмены в пользовательских приложениях можно найти во врезке "Отмена действия процедуры VBA".

В утилите Text Tools отмена ранее выполненной операции осуществляется путем предварительного сохранения исходных данных в рабочем листе. Как только пользователь отменяет операцию, сохраненные ранее данные копируются обратно в рабочую книгу.

В утилите Text Tools в модуле `Module1` общедоступные данные объявляются следующим образом.

```
Public UndoRange As Range
Public UserSelection As Range
```

Перед изменением данных процедура `ApplyButtonClick` вызывает процедуру `SaveForUndo`. В начале процедуры определяются три переменные.

```
Set UserSelection = Selection
Set UndoRange = WorkRange
ThisWorkbook.Sheets(1).UsedRange.Clear
```

Объектная переменная `UserSelection` сохраняет область, выделенную пользователем, благодаря чему можно отменить выделение области в дальнейшем. В данном случае `WorkRange` — это объект `Range`, который включает рабочий диапазон без пустых ячеек и ячеек с формулами. Третий оператор удаляет любые сохраненные данные.

После этого выполняется цикл.

```
For Each RngArea In WorkRange.Areas
   ThisWorkbook.Sheets(1).Range _
      (RngArea.Address).Formula = RngArea.Formula
Next RngArea
```

В цикле просматривается содержимое объекта `WorkRange`, а его данные сохраняются в рабочем листе (если этот объект состоит только из одного непрерывного диапазона, то в нем анализируется всего одна область).

После успешного выполнения описанных выше операций в коде вызывается метод `OnUndo`. С его помощью определяется процедура отмены выполненных действий. Например, в случае изменения регистра текста этот оператор принимает следующий вид.

```
Application.OnUndo "Отмена изменения регистра", "UndoTextTools"
```

Раскрывающийся список **Отменить** в Excel дополнится новой командой: **Отменить изменение регистра** (рис. 16.6). После ее выбора выполняется процедура `UndoTextTools`, приведенная ниже.

```
rivate Sub UndoTextTools()
  Отмена последней операции
  Dim a As Range
  On Error GoTo ErrHandler
  Application.ScreenUpdating = False
  With UserSelection
     .Parent.Parent.Activate
     .Parent.Activate
     .Select
  End With
  For Each a In UndoRange.Areas
     a.Formula = ThisWorkbook.Sheets(1).Range (a.Address).Formula
  Next a
  Application.ScreenUpdating = True
  On Error GoTo 0
  Exit Sub
rrHandler:
  Application.ScreenUpdating = True
  MsgBox "Невозможно отменить операцию", vbInformation, APPNAME
  On Error GoTo 0
nd Sub
```

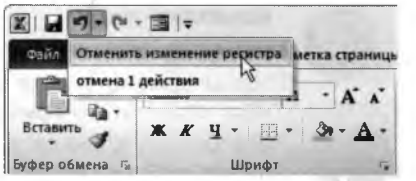


Рис. 16.6. Утилита TextTools допускает единственный уровень отмены

В процедуре `UndoTextTools` сначала проверяется, активизированы ли нужные рабочая книга и рабочий лист, а затем выбирается исходный диапазон, выделенный пользователем. Затем выполняется цикл по каждой области, в которой находятся сохраненные данные (ее доступность обеспечивается с помощью глобальной переменной `UndoRange`), а данные копируются обратно в их исходное место (при этом переписываются измененные данные).

Отмена действия процедуры VBA

Пользователи уже привыкли к возможности отмены ранее выполненных операций. Так, например, практически любая выполненная в Excel операция может быть отменена. Более того, начиная с версии в Excel 2007 количество уровней отмены возросло с 16 до 100.

В процессе программирования на VBA у вас может возникнуть вопрос о том, можно ли отменить результат выполнения процедуры. Ответ на него положительный, хотя на практике добиться этого непросто.

Отмена результата выполнения процедуры VBA не реализуется автоматически. Процедура должна в каком-то месте сохранять сведения о предыдущем состоянии системы, чтобы его можно было восстановить после выбора команды Отменить (Undo) — на панели быстрого доступа. Практическая реализация этого подхода зависит от действий, выполняемых процедурой. Старая информация может сохраняться на рабочем листе или в массиве. В крайнем случае вам придется сохранить весь рабочий лист. Например, если процедура модифицирует диапазон ячеек, нужно сохранять содержимое только этих ячеек.

Также имейте в виду, что в процессе выполнения процедуры VBA используется стек отмены Excel. Другими словами, после запуска макроса на выполнение предыдущие операции отменить невозможно.

Объект `Application` содержит метод `OnUndo`, который позволяет программисту указать текст, отображаемый в списке отмены, а также процедуру, которая вызывается после выбора пользователем команды отмены. Например, следующий оператор определяет команду Отменить выполнение макроса. После выбора пользователем команды Отменить⇨Отменить выполнение макроса вызывается процедура `UndoMyMacro`.

```
Application.OnUndo "Отменить выполнение макроса", "UndoMyMacro"
```

Компакт-диск

На прилагаемом к книге компакт-диске находится упрощенный пример, который демонстрирует симуляцию команды Отменить с помощью процедуры VBA. Файл этого примера называется `simple undo demo.xlsm` и предусматривает хранение данных в массиве, а не в рабочем листе. Массив имеет пользовательский тип данных и включает значения и адреса каждой ячейки.

Отображение файла справки

Я создал простой скомпилированный HTML-файл справки `texttools.chm`, который прилагается к этой утилите. После щелчка на кнопке справки (`HelpButton`), находящейся в диалоговом окне `UserForm`, вызывается следующая процедура.

```
Private Sub HelpButton_Click()
   Application.Help (ThisWorkbook.Path & "\" & "texttools.chm")
End Sub
```

На рис. 16.7 показан один из экранов справки.

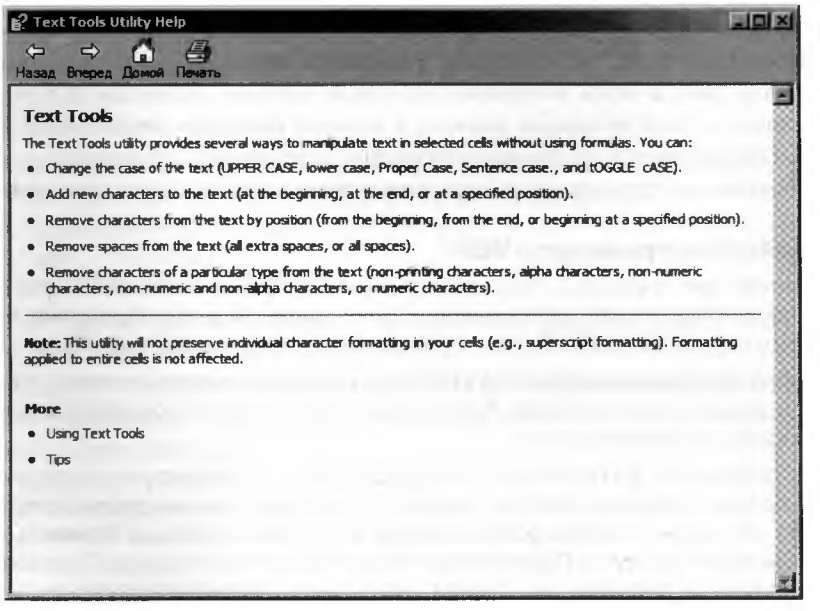


Рис. 16.7. Экран справки утилиты Text Tools

Компакт-диск

На прилагаемом к книге компакт-диске находятся все исходные файлы, которые были использованы для создания файла справки. Эти файлы можно найти в папке `\helpsource`. Если вы имеете смутное представление о файлах справки в формате HTML, обратитесь к главе 24 за дополнительными сведениями.

Добавление кода RibbonX

Последний этап в процессе создания нашей утилиты — определить способ ее запуска. До появления Excel 2007 операция вставки новой команды меню или кнопки панели инструментов не представляла особого труда. Однако с появлением "ленточного" пользовательского интерфейса все значительно усложнилось.

Для добавления кода RibbonX, используемого для создания группы и команды ленты, я воспользовался приложением Custom UI Editor для Microsoft Office. Это приложение не входит в состав Microsoft Office, но его можно легко найти в Интернете.

Перекрестная ссылка

Дополнительные сведения о работе с лентой и с приложением Custom UI Editor приведены в главе 22.

На рис. 16.8 показана часть ленты с новой группой (Utilities), добавленной в правой части вкладки Главная. Эта группа включает единственный элемент управления, который вызывает утилиту Text Tools.

Рис. 16.8. На вкладке ленты Главная появилась новая группа

```
Sub StartTextTools(control As IRibbonControl)
   Call ShowTextToolsDialog
End Sub
```

На рис. 16.9 показан код RibbonX, отображаемый в окне Custom UI Editor.

Рис. 16.9. Воспользуйтесь Custom UI Editor для определения способа вызова утилиты с ленты

Примечание

Если в состав рабочей ленты включена настроенная лента, ее настройки будут отображаться только в том случае, если рабочая книга активна. Но из этого правила есть исключение. Если настройки ленты включены в файл надстройки `XLAM` (как в рассматриваемом примере), изменения на ленте отображаются, если файл надстройки открыт, независимо от того, является ли книга активной.

Оценка проекта

В предыдущих разделах были описаны все компоненты утилиты Text Tools. Пришло время пересмотреть первоначальные цели проекта, чтобы оценить возможность их достижения. Ниже приведены первоначальные цели проекта с дополнительными комментариями.

- **Основные свойства утилиты перечислены в начале этой главы.** Цель достигнута.
- **Пользователю должна предоставляться возможность запрашивать перечисленные типы изменений по отношению и к текстовым, и к нетекстовым ячейкам.** Цель достигнута.
- **Утилита должна иметь тот же внешний вид и поведение, что и остальные команды Excel. Другими словами, утилита будет предоставлять пользователю диалоговое окно, которое выглядит так же, как и другие диалоговые окна Excel.** Как было отмечено ранее, утилита Text Tools незначительно отходит от этого принципа, так как вместо кнопки OK в ней используется кнопка Применить (Apply). Принимая во внимание критерий повышения удобства использования, на такое нарушение принципов поведения окон Excel можно согласиться.
- **Утилита должна быть реализована в форме надстройки и быть доступной с ленты.** Цель достигнута.
- **Утилита должна работать с текущим диапазоном ячеек (включая множественное выделение) и предоставлять пользователю возможность изменения диапазона ячеек, пока отображено диалоговое окно.** Цель достигнута. И поскольку скрывать диалоговое окно нет необходимости, элемент управления `RefEdit` не нужен.
- **Утилита должна помнить последнюю выполненную операцию, а также отображать текущие настройки при следующем открытии диалогового окна.** Цель достигнута (благодаря системному реестру Windows).
- **Утилита не должна обрабатывать ячейки с формулами.** Цель достигнута.
- **Утилита должна быть быстрой и эффективной. Например, если пользователь выбирает весь диапазон ячеек, пустые ячейки должны игнорироваться.** Цель достигнута.
- **Используемые диалоговые окна должны быть немодальными, в результате чего они отображаются на экране и всегда готовы к применению.** Цель достигнута.
- **Окно утилиты должно быть компактным, чтобы не перекрывать окно рабочего листа.** Цель достигнута.
- **Утилита обеспечивает возможность отмены выполненных пользователем изменений.** Цель достигнута.
- **Доступна исчерпывающая справка.** Цель достигнута.

Принципы работы утилиты

Если вы не совсем разобрались в принципах работы этой утилиты, загрузите рабочую книгу и воспользуйтесь отладчиком для пошагового просмотра кода. Целесообразно попытаться выполнить утилиту по отношению к различным выделенным диапазонам, включая весь рабочий лист. Вы заметите, что, независимо от размера диапазона, обрабатываются только текстовые ячейки, а пустые полностью игнорируются. Если на листе находится только одна текстовая ячейка, то утилита будет работать очень быстро независимо от того, выделена только эта ячейка или выделен весь рабочий лист.

Если файл надстройки преобразовать в стандартную рабочую книгу, можно увидеть, каким образом исходные данные (применяемые для выполнения отмены) хранятся в рабочем листе. Для выполнения такого преобразования дважды щелкните мышью на модуле кода `ЭтаКнига` (ThisWorkbook). Нажмите клавишу <F4> для отображения окна Properties (Свойства), после чего измените значение свойства `IsAddin` на `False`.

Дополнительно об утилитах Excel

Если вы всерьез заинтересовались разработкой утилит Excel, загрузите пробную версию пакета Power Utility Pack. В его состав входит более 60 полезных утилит (а также множество функций рабочего листа).

Помимо Power Utility Pack, в Интернете можно найти множество других полезных утилит Excel.

Глава 17

Работа со сводными таблицами

В этой главе...

- Вводный пример
- Создание сложных сводных таблиц
- Создание нескольких сводных таблиц
- Создание обратной сводной таблицы

Эта глава посвящена сводным таблицам Excel — одной из важнейших тем, которую просто невозможно обойти стороной.

Вводный пример

Сводные таблицы являются одним из самых мощных инновационных средств Excel. Они появились в Excel 5 и поддерживаются только в Excel (пока ни один программный продукт для управления электронными таблицами не предоставляет подобной возможности). В этой главе нет вступительной информации об использовании сводных таблиц. Предполагается, что вы уже знакомы с данной темой (как и с методами создания и изменения сводных таблиц вручную).

Создав сводную таблицу на основе базы данных или списка, можно быстро отобразить необходимые итоговые данные, которые другим способом представить на экране или бумаге просто невозможно. Кроме того, такое представление данных можно получить очень быстро. В Excel также поддерживается возможность создания кода VBA, который позволяет управлять сводными таблицами.

Этот раздел начинается с простого примера использования кода VBA для создания сводной таблицы.

На рис. 17.1 показана простая база данных на рабочем листе. Она состоит из четырех полей: `Представитель`, `Регион`, `Месяц` и `Продажи`. Записывается объем продаж определенного торгового представителя за определенный месяц.

	A	B	C	D	E
1	Представитель	Регион	Месяц	Продажи	
2	Эми	Север	Январь	33 488	
3	Эми	Север	Февраль	47 008	
4	Эми	Север	Март	32 128	
5	Боб	Север	Январь	34 736	
6	Боб	Север	Февраль	92 872	
7	Боб	Север	Март	76 128	
8	Чак	Юг	Январь	41 536	
9	Чак	Юг	Февраль	23 192	
10	Чак	Юг	Март	21 736	
11	Дут	Юг	Январь	44 834	
12	Дут	Юг	Февраль	32 002	
13	Дут	Юг	Март	23 932	
14					
15					
16					

Рис. 17.1. Эта таблица является хорошим кандидатом в сводные таблицы

Компакт-диск

Рассматриваемая далее рабочая книга `simple pivot table.xlsm` находится на прилагаемом компакт-диске.

Создание сводной таблицы

На рис. 17.2 показана сводная таблица, которая создана на основе указанных данных. Она обеспечивает отображение суммарного объема ежемесячных продаж каждого торгового представителя и содержит следующие поля.

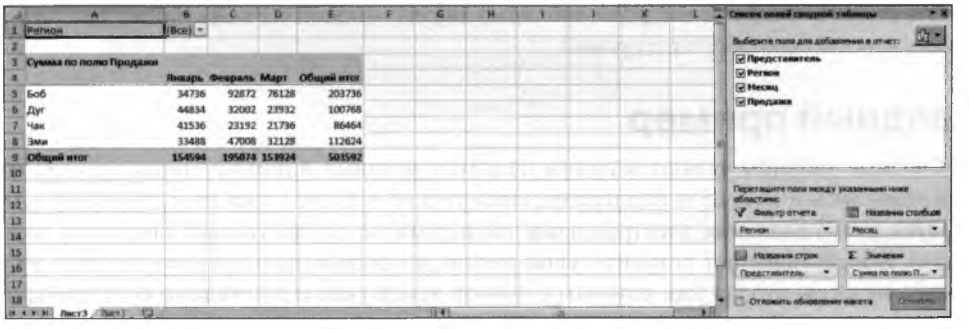


Рис. 17.2. Сводная таблица, созданная на основе таблицы, показанной на рис. 17.1

- **Регион.** Поле фильтра отчета в сводной таблице.
- **Представитель.** Поле строки в сводной таблице.
- **Месяц.** Поле столбца в сводной таблице.
- **Продажи.** Поле данных в сводной таблице, которое использует функцию `Sum` (СУММ).

Перед созданием сводной таблицы была включена функция записи макроса. Созданная сводная таблица размещается на новом рабочем листе. Далее представлен код, который был при этом сгенерирован.

```
Sub RecordedMacro()
   Range("A1").Select
   Sheets.Add
   ActiveWorkbook.PivotCaches.Create _
      (SourceType:=xlDatabase, _
      SourceData:="Лист1!R1C1:R13C4", _
      Version:=xlPivotTableVersion14).CreatePivotTable _
      TableDestination:="Лист2!R3C1", _
      TableName:="PivotTable1", _
      DefaultVersion:=xlPivotTableVersion14)
   Sheets("Лист2").Select
   Cells(3, 1).Select
   With ActiveSheet.PivotTables("PivotTable1") _
      .PivotFields("Представитель")
      .Orientation = xlRowField
      .Position = 1
   End With
   With ActiveSheet.PivotTables("PivotTable1") _
      .PivotFields("Месяц")
      .Orientation = xlColumnField
      .Position = 1
   End With
   ActiveSheet.PivotTables("PivotTable1") _
      .AddDataField ActiveSheet.PivotTables("PivotTable1") _
      .PivotFields("Продажи"), "Итого", xlSum
   With ActiveSheet.PivotTables("PivotTable1"). _
      PivotFields("Регион")
      .Orientation = xlPageField
      .Position = 1
   End With
End Sub
```

Запуск записанного макроса, скорее всего, приведет к ошибке. В процессе проверки кода вы обнаружите, что функция записи макроса "жестко закодировала" название рабочего листа (Лист2), на котором создается сводная таблица. Если подобный лист уже существует (либо добавляется новый лист с другим именем), выполнение макроса приведет к ошибке.

Несмотря на неработоспособность записанного макроса, вряд ли его можно назвать совсем бесполезным. Разработчики кода VBA для сводных таблиц найдут в этом коде немало полезного для себя.

Просмотр созданного кода

Код VBA, записанный при создании сводной таблицы, может привести вас в замешательство. Для того чтобы разобраться в записанном макросе, потребуется кое-что знать об используемых объектах (соответствующая информация содержится в интерактивном справочном руководстве).

- `PivotCaches` — коллекция объектов `PivotCache` в объекте `Workbook` (данные для сводной таблицы, которые хранятся в кэш-памяти сводных таблиц).
- `PivotTables` — коллекция объектов `PivotTable` в объекте `Worksheet`.
- `PivotFields` — коллекция полей в объекте `PivotTable`.
- `PivotItems` — коллекция отдельных элементов данных в поле.
- `CreatePivotTable` — метод объекта `PivotCache`, который создает сводную таблицу на основе данных, содержащихся в кэш-памяти.

Выбор данных для сводной таблицы

Данные, необходимые для создания сводной таблицы, должны быть в форме прямоугольной базы данных. Причем база данных может храниться как в виде диапазона листов (таблица либо обычный диапазон), так и в файле внешней базы данных. Несмотря на то что Excel может генерировать сводные таблицы на основе практически любой базы данных, нужно выполнить дополнительные условия.

Поля в таблице базы данных, на основе которой создается сводная таблица, относятся к следующим двум типам.

- **Данные (Data).** Включает суммируемые значения либо данные. Например, в случае с банковским счетом поле `Сумма` является полем данных.
- **Категория (Category).** Описывает данные. Например, в случае с тем же банковским счетом поля `Дата`, `Тип_счета`, `Открыто`, `Отделение` и `Владелец_счета` — это примеры полей категории, описывающие данные в поле `Сумма`.

Таблица базы данных, на основе которой может быть построена сводная таблица, должна быть нормализована. Иными словами, каждая запись (или строка) должна включать информацию, описывающую данные.

Единственная таблица базы данных может включать произвольное количество полей данных и категорий. При создании сводной таблицы обычно выполняется суммирование по одному или более полям данных. Ну а значения в полях категорий отображаются в сводной таблице в виде строк, столбцов либо фильтров.

Если изложенная выше концепция не вполне понятна, обратитесь к находящемуся на прилагаемом к книге компакт-диске файлу `normalized data.xlsx`. Эта рабочая книга содержит пример диапазона данных до и после выполнения нормализации, являющейся необходимым условием построения сводной таблицы.

Усовершенствование записанного кода сводной таблицы

Как и в случае с большинством записанных макросов, предыдущий пример не настолько эффективен, как следовало ожидать. Как уже отмечалось, его выполнение может завершиться ошибкой. Этот код можно упростить, чтобы сделать немного понятнее, а также исключить возможность появления ошибок. Ниже приведен код, написанный вручную, который создает ту же сводную таблицу, что и предыдущий макрос.

```
Sub CreatePivotTable()
   Dim PTCache As PivotCache
   Dim PT As PivotTable

'  Выделение кэш-памяти
   Set PTCache = ActiveWorkbook.PivotCaches.Create( _
      SourceType:=xlDatabase, _
      SourceData:=Range("A1").CurrentRegion)

'  Добавление нового листа в сводную таблицу
   Worksheets.Add

'  Создание сводной таблицы
   Set PT = ActiveSheet.PivotTables.Add( _
      PivotCache:=PTCache, _
      TableDestination:=Range("A3"))
```

```
' Определение полей
  With PT
     .PivotFields("Регион").Orientation = xlPageField
     .PivotFields("Месяц").Orientation = xlColumnField
     .PivotFields("Представитель").Orientation = xlRowField
     .PivotFields("Продажи").Orientation = xlDataField
     'заголовки полей отсутствуют
     .DisplayFieldCaptions = False
  End With
End Sub
```

В данном случае была упрощена процедура `CreatePivotTable` (что облегчило ее понимание) благодаря объявлению двух объектных переменных: `PTCache` и `PT`. Новый объект `PivotCache` был создан с помощью метода `Create`. Также был добавлен рабочий лист, который стал активным (на этом листе размещается сводная таблица). Затем был создан объект `PivotTable` с помощью метода `Add` из коллекции `PivotTables`. Последняя секция кода добавляет поля в сводную таблицу, а также указывает их положение в таблице путем присвоения значения свойству `Orientation`.

Обратите внимание, что исходный макрос жестко связан с диапазоном данных, на основе которого создается объект `PivotCache` (`'Лист1!R1C1:R13C4'`), и местоположением сводной таблицы (`Лист2`). В процедуре `CreatePivotTable` сводная таблица основана на текущем диапазоне, окружающем ячейку A1. Это гарантирует, что макрос будет выполняться даже тогда, когда в диапазон добавлены дополнительные данные.

Добавление рабочего листа до того, как была создана сводная таблица, исключает необходимость жесткого кодирования ссылки на лист. Еще одно отличие заключается в том, что написанный вручную макрос не определяет имя сводной таблицы. Да этого и не требуется, поскольку используется объектная переменная `PT`, которая выполняет эту задачу.

Совместимость сводных таблиц

Если планируется организовать общий доступ к рабочим книгам, включающим сводные таблицы, со стороны пользователей предыдущих версий Excel, уделите особое внимание вопросам совместимости. Во время просмотра кода записанного макроса (читайте раздел "Создание сводной таблицы") нетрудно заметить следующую инструкцию:

```
DefaultVersion:=xlPivotTableVersion14
```

Если рабочая книга находится в режиме совместимости, на ее месте будет такая инструкция:

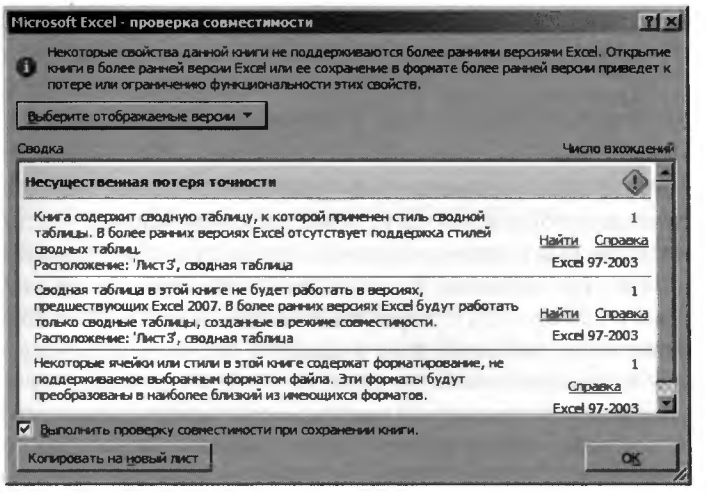
```
DefaultVersion:=xlPivotTableVersion10
```

Также вы отметите различия в коде записанного макроса, причина которых заключается в изменениях сводных таблиц, начиная с версии Excel 2007.

Предположим, что в среде Excel 2010 создана рабочая книга, которая была передана пользователям Excel 2003. Они увидят таблицу, но не смогут ее обновить. Другими словами, они получат в свое распоряжение статическую таблицу, включающую ряд числовых показателей.

Для обеспечения обратной совместимости сводных таблиц в Excel 2010 следует сначала сохранять файлы в формате `XLS`, а затем повторно их открывать. В этом случае обеспечивается работоспособность сводных таблиц при открытии содержащих их документов в версиях, предшествующих Excel 2007. Но в этом случае вы не сможете воспользоваться новыми свойствами сводных таблиц, которые появились в версиях Excel 2007 и Excel 2010.

К счастью, существует модуль проверки совместимости Excel, который может выявить проблемы, связанные с совместимостью (см. иллюстрацию). Но этот модуль не в состоянии проверить поддержку обратной совместимости кода, генерируемого макросом.



Рассматриваемый в этой главе макрос не в состоянии генерировать сводные таблицы, поддерживающие обратную совместимость.

Примечание

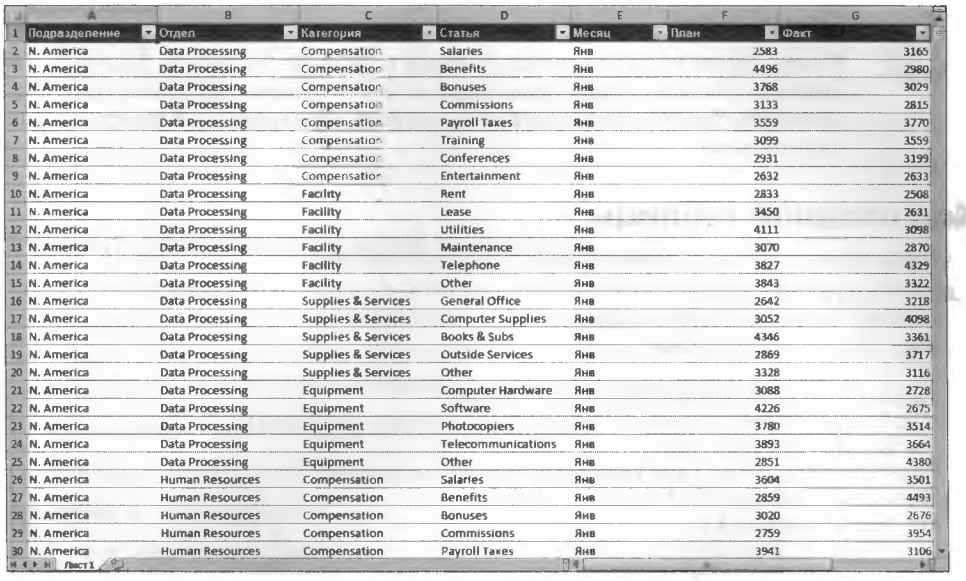
Данный код можно сделать более универсальным, используя индексы вместо названий в коллекции `PivotFields`. При этом, если пользователь изменяет заголовки столбцов, код остается работоспособным. Например, в более универсальном макросе используется оператор `PivotFields(1)` вместо оператора `PivotFields('Регион')`.

Чтобы лучше понять рассматриваемую тему, запишите собственный макрос и изучите ключевые объекты, методы и свойства. После этого обратитесь к разделам справочного руководства, чтобы разобраться, как все это работает. Практически во всех случаях необходимо вносить изменения в "свежеиспеченный" макрос. Как только вам станут понятны принципы управления сводными таблицами, можете приступать к созданию кода без предварительной записи макроса.

Создание сложных сводных таблиц

В этом разделе будет представлен код VBA, используемый для создания сравнительно сложной сводной таблицы.

На рис. 17.3 показан фрагмент базы данных на листе. В этой таблице содержится 15840 строк иерархически упорядоченной информации о бюджете корпорации. В корпорации существует пять подразделений; каждое подразделение содержит одиннадцать отделов. Отдел имеет четыре бюджетные категории, а каждая категория состоит из нескольких пунктов. Бюджетные и фактические расходы указываются для каждого (из двенадцати) месяца. Цель — создать сводную таблицу, которая суммирует эту информацию.



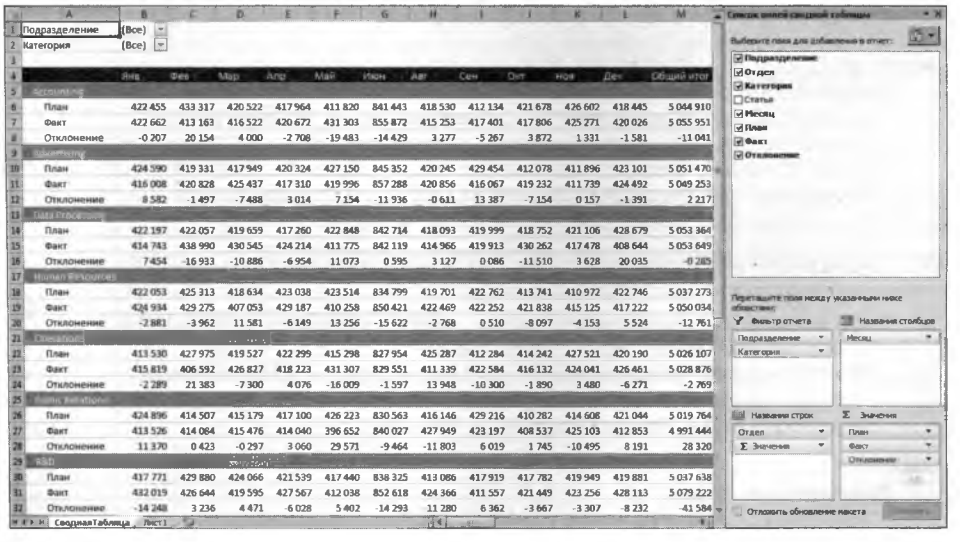
	A	B	C	D	E	F	G
1	Подразделение	Отдел	Категория	Статья	Месяц	План	Факт
2	N. America	Data Processing	Compensation	Salaries	Янв	2583	3165
3	N. America	Data Processing	Compensation	Benefits	Янв	4496	2980
4	N. America	Data Processing	Compensation	Bonuses	Янв	3768	3029
5	N. America	Data Processing	Compensation	Commissions	Янв	3133	2815
6	N. America	Data Processing	Compensation	Payroll Taxes	Янв	3559	3770
7	N. America	Data Processing	Compensation	Training	Янв	3099	3559
8	N. America	Data Processing	Compensation	Conferences	Янв	2931	3199
9	N. America	Data Processing	Compensation	Entertainment	Янв	2632	2633
10	N. America	Data Processing	Facility	Rent	Янв	2833	2508
11	N. America	Data Processing	Facility	Lease	Янв	3450	2631
12	N. America	Data Processing	Facility	Utilities	Янв	4111	3098
13	N. America	Data Processing	Facility	Maintenance	Янв	3070	2870
14	N. America	Data Processing	Facility	Telephone	Янв	3827	4329
15	N. America	Data Processing	Facility	Other	Янв	3843	3322
16	N. America	Data Processing	Supplies & Services	General Office	Янв	2642	3218
17	N. America	Data Processing	Supplies & Services	Computer Supplies	Янв	3052	4098
18	N. America	Data Processing	Supplies & Services	Books & Subs	Янв	4346	3361
19	N. America	Data Processing	Supplies & Services	Outside Services	Янв	2869	3717
20	N. America	Data Processing	Supplies & Services	Other	Янв	3328	3116
21	N. America	Data Processing	Equipment	Computer Hardware	Янв	3088	2728
22	N. America	Data Processing	Equipment	Software	Янв	4226	2675
23	N. America	Data Processing	Equipment	Photocopiers	Янв	3780	3514
24	N. America	Data Processing	Equipment	Telecommunications	Янв	3893	3664
25	N. America	Data Processing	Equipment	Other	Янв	2851	4380
26	N. America	Human Resources	Compensation	Salaries	Янв	3604	3501
27	N. America	Human Resources	Compensation	Benefits	Янв	2859	4493
28	N. America	Human Resources	Compensation	Bonuses	Янв	3020	2676
29	N. America	Human Resources	Compensation	Commissions	Янв	2759	3954
30	N. America	Human Resources	Compensation	Payroll Taxes	Янв	3941	3106

Рис. 17.3. Данные из этой рабочей книги будут собраны в сводной таблице

Компакт-диск

Рассматриваемая в данном разделе рабочая книга находится на прилагаемом компакт-диске в файле `budget pivot table.xlsm`.

На рис. 17.4 показана сводная таблица, созданная на основе приведенных выше данных. Обратите внимание на то, что она содержит вычисляемое поле, которое называется `Отклонение`. Значение этого поля представляет собой разницу между значениями полей `План` и `Факт`.



	A	B
1	Подразделение	(Все)
2	Категория	(Все)

	Янв	Фев	Мар	Апр	Май	Июн	Авг	Сен	Окт	Ноя	Дек	Общий итог
Accounting												
План	422 455	433 317	420 522	417 964	411 820	841 443	418 530	412 134	421 678	426 602	418 445	5 044 910
Факт	422 662	413 163	416 522	420 672	431 303	855 872	415 253	417 401	417 806	425 271	420 026	5 055 951
Отклонение	-0 207	20 154	4 000	-2 708	-19 483	-14 429	3 277	-5 267	3 872	1 331	-1 581	-11 041
Advertising												
План	424 590	419 331	417 949	420 324	427 150	845 352	420 245	429 454	412 078	411 896	423 101	5 051 470
Факт	416 008	420 828	425 437	417 310	419 996	857 288	420 856	416 067	419 232	411 739	424 492	5 049 253
Отклонение	8 582	-1 497	-7 488	3 014	7 154	-11 936	-0 611	13 387	-7 154	0 157	-1 391	2 217
Data Processing												
План	422 197	422 057	419 659	417 260	422 848	842 714	418 093	419 999	418 752	421 106	428 679	5 053 364
Факт	414 743	438 990	430 545	424 214	411 775	842 119	414 966	419 913	430 262	417 478	408 644	5 053 649
Отклонение	7 454	-16 933	-10 886	-6 954	11 073	0 595	3 127	0 086	-11 510	3 628	20 035	-0 285
Human Resources												
План	422 053	425 313	418 634	423 038	423 514	834 799	419 701	422 762	413 741	410 972	422 746	5 037 273
Факт	424 934	429 275	407 053	429 187	410 258	850 421	422 469	422 252	421 838	415 125	417 222	5 050 034
Отклонение	-2 881	-3 962	11 581	-6 149	13 256	-15 622	-2 768	0 510	-8 097	-4 153	5 524	-12 761
Operations												
План	413 530	427 975	419 527	422 299	415 298	827 954	425 287	412 284	414 242	427 521	420 190	5 026 107
Факт	415 819	406 592	426 827	418 223	431 307	829 551	411 339	422 584	416 132	424 041	426 461	5 028 876
Отклонение	-2 289	21 383	-7 300	4 076	-16 009	-1 597	13 948	-10 300	-1 890	3 480	-6 271	-2 769
Public Relations												
План	424 896	414 507	415 179	417 100	426 223	830 563	416 146	429 216	410 282	414 608	421 044	5 019 764
Факт	413 526	414 084	415 476	414 040	396 652	840 027	427 949	423 197	408 537	425 103	412 853	4 991 444
Отклонение	11 370	0 423	-0 297	3 060	29 571	-9 464	-11 803	6 019	1 745	-10 495	8 191	28 320
R&D												
План	417 771	429 880	424 066	421 539	417 440	838 325	413 086	417 919	417 782	419 949	419 881	5 037 638
Факт	432 019	426 644	419 595	427 567	412 038	852 618	424 366	411 557	421 449	423 256	428 113	5 079 222
Отклонение	-14 248	3 236	4 471	-6 028	5 402	-14 293	11 280	6 362	-3 667	-3 307	-8 232	-41 584

Рис. 17.4. Сводная таблица, созданная на основе данных из рис. 17.3

Примечание

Можно также вставить новую колонку в таблицу и создать формулу, вычисляющую разницу между плановыми и фактическими показателями. Эта возможность будет недоступной в том случае, если данные берутся из внешнего источника.

Код сводной таблицы

Ниже приводится код, генерирующий сводную таблицу.

```
Sub CreatePivotTable()
   Dim PTcache As PivotCache
   Dim PT As PivotTable

   Application.ScreenUpdating = False
'  Удаление листа сводной таблицы (при его наличии)
   On Error Resume Next
   Application.DisplayAlerts = False
   Sheets("СводнаяТаблица").Delete
   On Error GoTo 0

'  Выделение кэш-памяти для сводной таблицы
   Set PTcache = ActiveWorkbook.PivotCaches.Create( _
      SourceType:=xlDatabase, _
      SourceData:=Range("A1").CurrentRegion.Address)

'  Добавление нового рабочего листа
   Worksheets.Add
   ActiveSheet.Name = "СводнаяТаблица"
   ActiveWindow.DisplayGridlines = False
'   Создание сводной таблицы на основе данных из кэш-памяти
   Set PT = ActiveSheet.PivotTables.Add( _
      PivotCache:=PTcache, _
      TableDestination:=Range("A1"), _
      TableName:="СводнаяТаблицаПлан")

   With PT
'     Добавление полей
      .PivotFields("Категория").Orientation = xlPageField
      .PivotFields("Подразделение").Orientation = xlPageField
      .PivotFields("Отдел").Orientation = xlRowField
      .PivotFields("Месяц").Orientation = xlColumnField
      .PivotFields("План").Orientation = xlDataField
      .PivotFields("Факт").Orientation = xlDataField
      .DataPivotField.Orientation = xlRowField

'     Добавление вычисляемого поля, определяющего отклонение
      .CalculatedFields.Add "Отклонение", "=План-Факт"
      .PivotFields("Отклонение").Orientation = xlDataField
'      Определение числового формата
      .DataBodyRange.NumberFormat = "0,000"

'     Применение стиля
      .TableStyle2 = "PivotStyleMedium2"

'     Сокрытие заголовков полей
      .DisplayFieldCaptions = False
```

```
    Изменение заглавий
    .PivotFields("Сумма по полю План").Caption = " План"
    .PivotFields("Сумма по полю Факт").Caption = " Факт"
    .PivotFields("Сумма по полю Отклонение").Caption = " Отклонение"
  End With
End Sub
```

Принцип работы сводной таблицы

Процедура `CreatePivotTable` начинает свою работу с удаления листа `СводнаяТаблица`, если он существует. После этого создается объект `PivotCache`, добавляется новый лист `СводнаяТаблица` и создается сводная таблица на основе объекта `PivotCache`. Далее программа добавляет поля к созданной сводной таблице:

- **Категория** — поле фильтра отчета (страницы);
- **Подразделение** — поле фильтра отчета (страницы);
- **Отдел** — поле строки;
- **Месяц** — поле столбца;
- **План** — поле данных;
- **Факт** — поле данных.

Обратите внимание, что свойство `Orientation` объекта `DataPivotField` установлено равным `xlRowField` с помощью следующего оператора:

```
.DataPivotField.Orientation = xlRowField
```

Этот оператор определяет общую ориентацию сводной таблицы, а также представляет ее с помощью поля **Значение суммы** в списке полей сводной таблицы (рис. 17.5). Попытайтесь переместить это поле в раздел **Надписи** столбцов, после чего посмотрите, как изменяется макет сводной таблицы.

Затем процедура использует метод `Add` из коллекции `CalculatedFields` для создания вычисляемого поля `Отклонение`, величина которого представляет собой результат вычитания значения поля `Факт` из значения поля `План`. Это вычисляемое поле относится к категории полей данных.

Примечание

Для добавления вычисляемого поля в сводную таблицу вручную воспользуйтесь командой Сводная таблица⇨Параметры⇨Вычисления⇨Поля, элементы и наборы⇨Вычисляемое поле (PivotTable⇨Options⇨Calculations⇨Fields, Items & Sets⇨Calculation Field) для перехода в диалоговое окно Вставка вычисляемого поля (Insert Calculated Field).

Также код выполняет небольшие "косметические" улучшения сводной таблицы:

- применяет числовой формат к объекту `DataBodyRange` (этот объект представляет данные всей сводной таблицы);
- применяет стиль;
- скрывает заголовки (эквивалентен элементу управления **Работа со сводными таблицами⇨Параметры⇨Показать/Скрыть⇨Заголовки полей** (PivotTable Toos⇨Options⇨Show/Hide⇨Field Headers);
- изменяет заголовки, отображенные в сводной таблице. Например, название `Сумма по полю План` (Sum of Budget) заменяется названием `План` (Budget). Обратите внимание, что перед названием `План` вставляется пробел. В Excel не допускается изменение заголовка, который соответствует имени поля, но, как видите, это ограничение можно обойти, добавив пробел.

Примечание

В процессе создания процедуры я постоянно пользовался функцией записи макросов в целях ознакомления с ее функциями. Также мне очень помогла интерактивная справочная система (вместе с многочисленными тестами и поучительными ошибками).

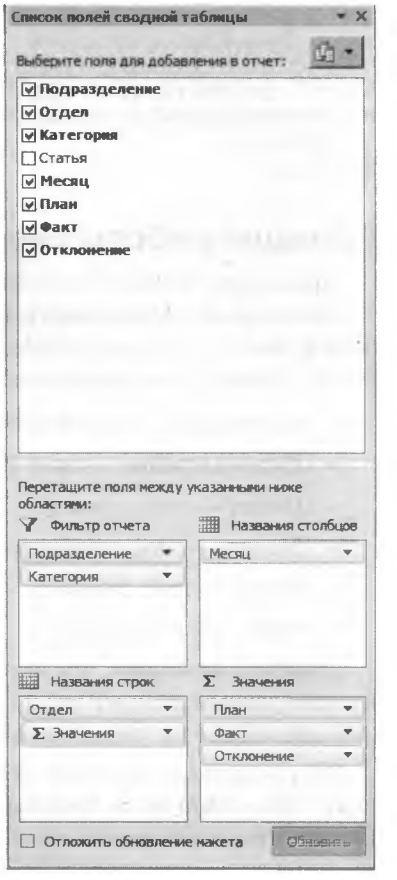


Рис. 17.5. Список полей сводной таблицы

Создание нескольких сводных таблиц

При выполнении данного упражнения создается целая серия сводных таблиц, в которых суммируются данные, собранные в процессе опроса покупателей. Эти данные находятся в базе данных рабочих листов (рис. 17.6), которая состоит из 150 строк. Каждая строка включает указание пола респондента, а также числовой рейтинг, который изменяется от 1 до 5 по каждому из 14 пунктов опроса.

Компакт-диск

Эта рабочая книга находится на прилагаемом компакт-диске в файле `survey data pivot tables.xlsm`.

На рис. 17.7 показаны некоторые из 28 сводных таблиц, созданные макросом. Суммирование по каждому элементу опроса производилось с помощью двух сводных таблиц (в одной отображаются значения фактической частоты, в другой — процентные соотношения).

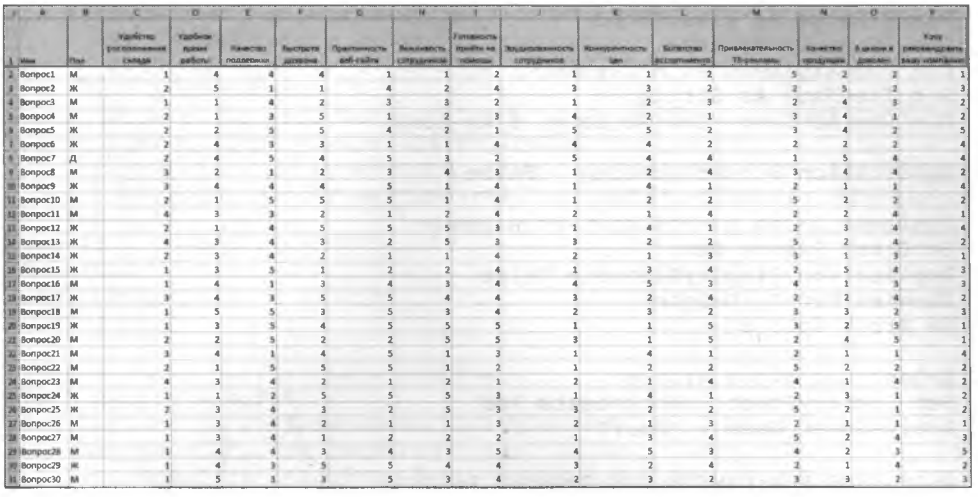


Рис. 17.6. Создание набора сводных таблиц, суммирующих данные опроса

Удобство расположения склада

Частота				
	Д	Ж	М	Общий итог
Решительно не согласен		28	40	68
Не согласен	1	19	16	36
Колеблюсь		15	9	24
Согласен		6	14	20
Полностью согласен		2		2
Общий итог	**1**	**70**	**79**	**150**

Удобство расположения склада

Процент				
	Д	Ж	М	Общий итог
Решительно не согласен	0,0%	40,0%	46%	45,3%
Не согласен	100,0%	27,1%	20,3%	24,0%
Колеблюсь	0,0%	21,4%	11,4%	16,0%
Согласен	0,0%	8,6%	17,7%	13,3%
Полностью согласен	0,0%	2,9%	0,0%	1,3%

Удобное время работы

Частота				
	Д	Ж	М	Общий итог
Решительно не согласен		11	13	24
Не согласен		7	11	18
Колеблюсь		30	26	56
Согласен	1	19	22	42
Полностью согласен		3	7	10
Общий итог	**1**	**70**	**79**	**150**

Удобное время работы

Процент				
	Д	Ж	М	Общий итог
Решительно не согласен	0,0%	15,7%	16,5%	16,0%
Не согласен	0,0%	10,0%	13,9%	12,0%
Колеблюсь	0,0%	42,9%	32,9%	37,3%
Согласен	100,0%	27,1%	27,8%	28,0%
Полностью согласен	0,0%	4,3%	8,9%	6,7%

Качество поддержки

Частота				
	Д	Ж	М	Общий итог
Решительно не согласен		7	14	21
Не согласен		7	4	11
Колеблюсь		16	14	30
Согласен		29	29	58
Полностью согласен	1	11	18	30
Общий итог	**1**	**70**	**79**	**150**

Качество поддержки

Процент				
	Д	Ж	М	Общий итог
Решительно не согласен	0,0%	10,0%	17,7%	14,0%
Не согласен	0,0%	10,0%	5,1%	7,3%
Колеблюсь	0,0%	22,9%	17,7%	20,0%
Согласен	0,0%	41,4%	36,7%	38,7%
Полностью согласен	100,0%	15,7%	22,8%	20,0%

Рис. 17.7. Шесть из 28 сводных таблиц, созданных с помощью процедуры VBA

Ниже приводится процедура VBA, с помощью которой создавались сводные таблицы.

```
Sub MakePivotTables()
' Эта процедура создает 28 сводных таблиц
```

```
    Dim PTCache As PivotCache
    Dim PT As PivotTable
    Dim SummarySheet As Worksheet
    Dim ItemName As String
    Dim Row As Long, Col As Long, i As Long

    Application.ScreenUpdating = False

'   Удаление листа итогов при его наличии
    On Error Resume Next
    Application.DisplayAlerts = False
    Sheets("Итог").Delete
    On Error GoTo 0

'   Добавление листа итогов
    Set SummarySheet = Worksheets.Add
    ActiveSheet.Name = "Итог"

'   Выделение кэш-памяти для сводной таблицы
    Set PTCache = ActiveWorkbook.PivotCaches.Create( _
       SourceType:=xlDatabase, _
       SourceData:=Sheets("ДанныеОпроса").Range("A1"). _
       CurrentRegion)

    Row = 1
       For i = 1 To 14
          For Col = 1 To 6 Step 5 '2 колонки
             ItemName = Sheets("ДанныеОпроса").Cells(1, i + 2)
             With Cells(Row, Col)
                .Value = ItemName
                .Font.Size = 16
             End With
'         Создание сводной таблицы
          Set PT = ActiveSheet.PivotTables.Add( _
             PivotCache:=PTCache, _
             TableDestination:=SummarySheet.Cells(Row + 1, Col))

'         Добавление полей
          If Col = 1 Then ' таблицы частот
          With PT.PivotFields(ItemName)
             .Orientation = xlDataField
             .Name = "Частота"
             .Function = xlCount
          End With
          Else ' процентные таблицы
          With PT.PivotFields(ItemName)
             .Orientation = xlDataField
             .Name = "Процент"
             .Function = xlCount
             .Calculation = xlPercentOfColumn
             .NumberFormat = "0.0%"
          End With
          End If
          PT.PivotFields(ItemName).Orientation = xlRowField
          PT.PivotFields("Пол").Orientation = xlColumnField
          PT.TableStyle2 = "PivotStyleMedium2"
          PT.DisplayFieldCaptions = False
          If Col = 6 Then
```

```
'        Добавление шкал данных в последнюю колонку
         PT.ColumnGrand = False
         PT.DataBodyRange.Columns(3).FormatConditions. _
            AddDatabar
         With pt.DataBodyRange.Columns(3).FormatConditions(1)
         .BarFillType = xlDataBarFillSolid
         .MinPoint.Modify newtype:=xlConditionValueNumber, newvalue:=0
         .MaxPoint.Modify newtype:=xlConditionValueNumber, newvalue:=1
      End With
   End If
   Next Col
   Row = Row + 10
   Next i

'  Замена чисел описательным текстом
   With Range("A:A,F:F")
      .Replace "1", "Решительно не согласен"
      .Replace "2", "Не согласен"
      .Replace "3", "Колеблюсь"
      .Replace "4", "Согласен"
      .Replace "5", "Полностью согласен"
   End With
End Sub
```

Обратите внимание на то, что все сводные таблицы были созданы из единственного объекта `PivotCache`.

Сводные таблицы создаются с помощью вложенных циклов. Счетчик цикла `Col` изменяется от 1 до 6 с помощью параметра `Step`. Для второго столбца сводных таблиц немного изменяется характер обработки. Выполняются следующие действия:

- отображается величина (в виде процента от значения в столбце);
- не отображаются окончательные итоги для строк;
- используется числовой формат;
- отображаются цветовые шкалы, с помощью которых реализуется условное форматирование.

Переменная `Row` отслеживает начальную строку в каждой сводной таблице. Завершающая операция заключается в замене числовых категорий в столбцах A и F текстом. Например, вместо единицы подставляется текст "Полностью согласен".

Создание обратной сводной таблицы

Сводная таблица представляет собой результат суммирования данных в обычной таблице. А как быть в случае, если у вас имеется итоговая таблица, на основе которой нужно воссоздать исходную таблицу? Соответствующий пример приводится на рис. 17.8. Диапазон ячеек B2:F14 представляет собой итоговую таблицу — упрощенный вариант сводной таблицы. В столбцах I:K находится состоящая из 48 строк таблица, созданная на основе итоговой таблицы. В этой таблице каждая строка содержит точку данных, а первые два столбца включают описание этой точки.

В Excel не существует прямого способа преобразования итоговой таблицы в обычную, но эту работу может выполнить макрос VBA. После его создания я разработал диалоговое окно `UserForm`, показанное на рис. 17.9. В нем определяются входной и выходной диапазоны, а также имеется опция преобразования выходного диапазона в таблицу.

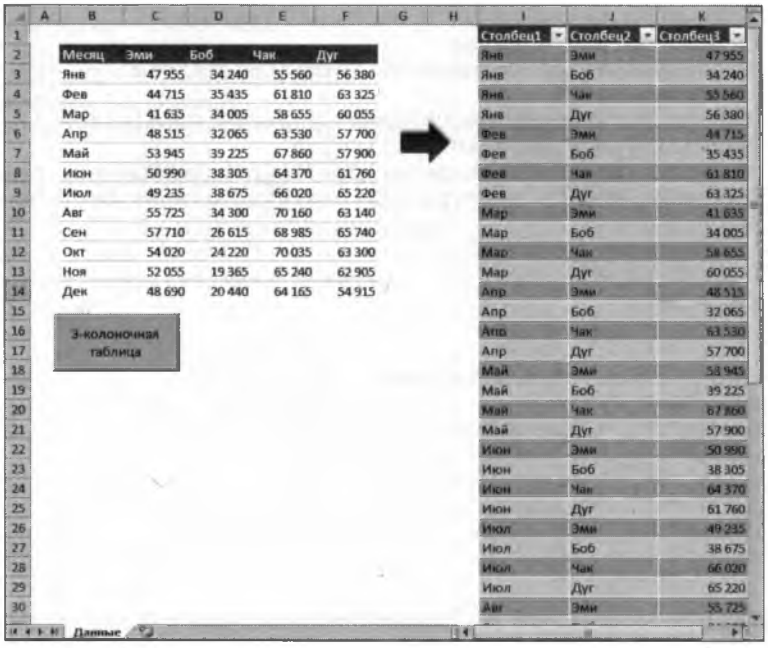


Рис. 17.8. Итоговая таблица (слева) может быть преобразована в обычную таблицу (справа)

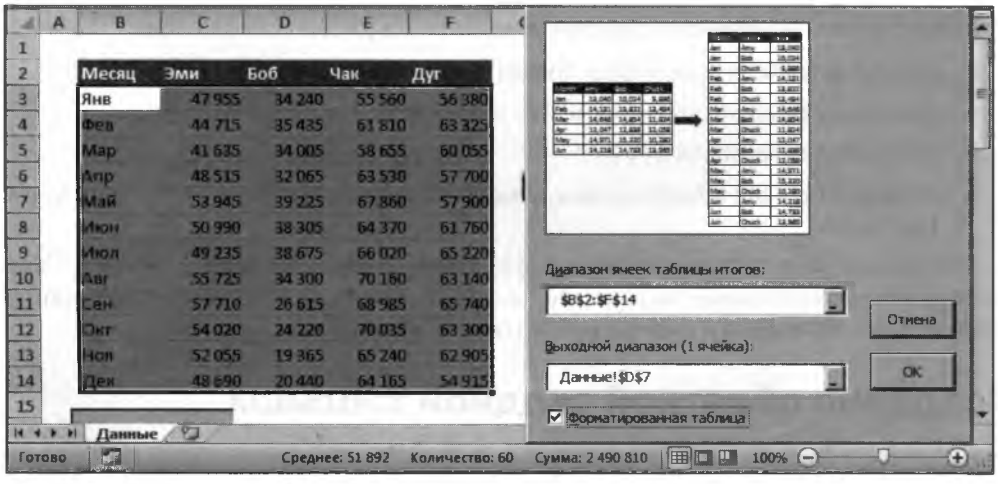


Рис. 17.9. В этом диалоговом окне пользователь может определить диапазоны

Компакт-диск

Эта рабочая книга находится на прилагаемом компакт-диске в файле `reverse pivot table.xlsm`.

После щелчка мышью на кнопке OK в диалоговом окне `UserForm` код VBA проверяет диапазоны и вызывает процедуру `ReversePivot` с помощью следующей инструкции:

```
Call ReversePivot(SummaryTable, OutputRange, cbCreateTable)
```

Этой процедуре передаются следующие три аргумента.

- SummaryTable. Объект Range, который представляет итоговую таблицу.
- OutputRange. Объект Range, представляющий верхнюю левую ячейку выходного диапазона.
- cbCreateTable. Флажок (объект Checkbox) в окне UserForm.

Эта процедура работает с итоговой таблицей любого размера. Количество строк данных в выходной таблице можно подсчитать по формуле (r-1) * (c-1), где переменные r и c представляют собой количество строк и столбцов в итоговой таблице соответственно.

Ниже приведен код процедуры ReversePivot.

```
Sub ReversePivot(SummaryTable As Range, _
   OutputRange As Range, CreateTable As Boolean)
   Dim r As Long, c As Long
   Dim OutRow As Long, OutCol As Long
'  Преобразование диапазона
   OutRow = 2
   Application.ScreenUpdating = False
   OutputRange.Range("A1:C3") = Array("Столбец", "Столбец2", _
                                      "Столбец3")
   For r = 2 To SummaryTable.Rows.Count
      For c = 2 To SummaryTable.Columns.Count
         OutputRange.Cells(OutRow, 1) = SummaryTable.Cells(r, 1)
         OutputRange.Cells(OutRow, 2) = SummaryTable.Cells(1, c)
         OutputRange.Cells(OutRow, 3) = SummaryTable.Cells(r, c)
         OutRow = OutRow + 1
      Next c
   Next r
'  Создать таблицу?
   If CreateTable Then _
      ActiveSheet.ListObjects.Add xlSrcRange, _
      OutputRange.CurrentRegion, , xlYes
End Sub
```

Описанная процедура достаточно проста. Код выполняет циклический обход строк и столбцов во входном диапазоне, а затем записывает данные в выходной диапазон, который всегда состоит из трех столбцов. Переменная OutRow отслеживает текущую строку в выходном диапазоне. Если пользователь устанавливает флажок, выходной диапазон преобразуется в таблицу. При этом используется метод Add из коллекции ListObjects.

Глава 18

Управление диаграммами

В этой главе...

- Кратко о диаграммах
- Создание внедренной диаграммы
- Размещение диаграммы на листе диаграммы
- Активизация диаграммы с помощью кода VBA
- Перемещение диаграммы
- Деактивизация диаграммы
- Определение активности диаграммы
- Удаление объектов из коллекции `ChartObjects` или `Charts`
- Циклический просмотр диаграмм
- Изменение размеров и выравнивание диаграмм
- Экспорт диаграммы
- Изменение применяемых в диаграмме данных
- Отображение подписей для данных на диаграмме
- Отображение диаграммы в пользовательском диалоговом окне
- События диаграмм
- Тонкости создания диаграмм
- Анимирование диаграмм
- Создание интерактивной диаграммы без написания макросов
- Спарклайны

Данная глава посвящена созданию в Excel самых различных диаграмм.

Кратко о диаграммах

В Excel можно создавать самые разные диаграммы на основе данных рабочего листа. Причем пользователь может управлять практически каждым их элементом.

Диаграммы Excel насыщены объектами, каждый из которых имеет собственные свойства и методы. Таким образом, управление диаграммами посредством кода VBA связано с определенными трудностями, а функция записи макросов в Excel не слишком помогает при решении возникающих проблем. В этой главе рассматриваются основные концепции создания кода VBA, предназначенного для управления диаграммами. Главное — разобраться в тонкостях объектной модели диаграмм. Для начала ознакомимся с общими сведениями о диаграммах Excel.

Новинка

В Excel 2010 появился новый вид диаграмм, которые называются спарклайнами. Этот вид диаграмм имеет настолько маленькие размеры, что помещается в одной ячейке. При создании спарклайнов используется объектная модель, которая коренным образом отличается от объектной модели, применяемой при создании диаграмм. Эта тема подробнее рассматривается далее.

Расположение диаграмм

В Excel диаграмма может размещаться в одном из следующих двух мест рабочей книги.

- **На рабочем листе в качестве встроенного объекта.** На рабочем листе может находиться произвольное количество встроенных диаграмм.
- **На отдельном листе диаграммы.** Обычно на листе диаграммы находится одна диаграмма.

Большинство диаграмм создается вручную посредством команд из группы Вставка⇨Диаграммы (Insert⇨Charts). Для создания диаграмм можно также прибегнуть к коду VBA. Он может применяться для изменения существующих диаграмм.

Совет

Самый быстрый способ создания диаграммы — выделить данные и нажать комбинацию клавиш <Alt+F1>. В результате выполнения этих действий Excel создает внедренную диаграмму, а также выбирает ее тип, заданный по умолчанию. Для создания диаграммы на листе диаграммы выделите данные и нажмите клавишу <F11>.

Ключевой при работе с диаграммами является концепция *активной диаграммы*. Когда пользователь щелкает на встроенной диаграмме или выделяет лист диаграммы, активизируется объект `Chart`. В VBA свойство `ActiveChart` возвращает активный объект `Chart` (если такой существует). Можно создать код, который будет управлять этим объектом (он будет подобен коду для работы с объектом `Workbook`, возвращаемым свойством `ActiveWorkbook`).

Ниже приведен пример. Если диаграмма активизирована, то следующий оператор отобразит свойство `Name` активного объекта `Chart`:

```
MsgBox ActiveChart.Name
```

Если диаграмма не активизирована, выполнение предыдущего оператора приведет к появлению ошибки.

Примечание

Вовсе не обязательно активизировать диаграмму для того, чтобы управлять ею с помощью кода VBA.

Диаграммы и функция записи макроса

В предыдущих главах неоднократно рекомендовалось использовать функцию записи макросов для изучения объектов, свойств и методов. В издании этой книги, посвященном версии Excel 2007, упоминалось о серьезной проблеме, связанной с применением функции записи макросов при работе с диаграммами, а именно: эту функцию невозможно было применять для записи каких-либо операций, связанных с созданием либо изменением диаграмм. К счастью, эта проблема была устранена в версии Excel 2010. Да-да, функция записи макросов в Excel 2010 работает вполне удовлетворительно. Сгенерированный ею код, конечно, далек от совершенства, но все же это большой шаг вперед по сравнению с версией Excel 2007.

Автоматически сгенерированные макросы лучше всего использовать в учебных целях. Код подобного макроса поможет установить взаимосвязь между объектами, свойствами и методами.

Замечание о совместимости

Код VBA, представленный в этой главе, использует ряд новых свойств и методов, применяемых для управления диаграммами, которые появились в Excel 2007. Поэтому этот код вряд ли будет работать в предыдущих версиях Excel.

Объектная модель диаграммы

Первое знакомство с объектной моделью диаграммы может вызвать у пользователя чувство растерянности. И это неудивительно, ведь структура этой модели очень сложна.

Предположим, что необходимо изменить заголовок, отображаемый на встроенной диаграмме. Объект верхнего уровня — это объект `Application` (Excel). Объект `Application` содержит объект `Workbook`, в котором находится объект `Worksheet`. Последний содержит объект `ChartObject`, а в нем расположен объект `Chart`. Объект `Chart` содержит объект `ChartTitle`, а объект `ChartTitle` имеет свойство `Text`, которое представляет объект, отображаемый в заголовке диаграммы. Ниже в графическом виде представлена иерархия объектов встроенной диаграммы.

```
Application
   Workbook
      Worksheet
         ChartObject
            Chart
               ChartTitle
```

Код VBA должен, конечно же, следовать этой объектной модели. Например, чтобы присвоить заголовку диаграммы значение "Ежегодные продажи", воспользуйтесь следующим оператором VBA.

```
WorkSheets("Лист1").ChartObjects(1).Chart.ChartTitle. _
   .Text = "Ежегодные продажи"
```

В этом выражении предполагается, что активная рабочая книга представлена объектом `Workbook`. Оператор обращается к первому элементу коллекции рабочих листов

(ChartObjects) под названием Лист1. Свойство Chart возвращает объект Chart, а свойство ChartTitle возвращает объект ChartTitle. В самом конце оператора вы обращаетесь к свойству Text.

Обратите внимание, что указанный выше оператор не будет работать в случае, если диаграмма не имеет заголовка. Для добавления к диаграмме заголовка, заданного по умолчанию (отображает текст Заголовок диаграммы (Chart Title)), воспользуйтесь следующим оператором:

```
Worksheets("Лист1").ChartObjects(1).Chart.HasTitle = True
```

Для диаграммы на листе диаграммы объектная модель несколько отличается, так как она не содержит объект Worksheet или ChartObject. Например, ниже представлена объектная модель для диаграммы на листе диаграммы.

```
Application
   Workbook
      Chart
         ChartTitle
```

Следующий оператор можно использовать для просмотра заголовка диаграммы "Ежегодные продажи":

```
Sheets("Лист1").ChartTitle.Text = "Ежегодные продажи"
```

Другими словами, лист диаграммы является объектом Chart и не содержит объект ChartObject. Если рассмотреть ситуацию в общем, то легко понять, что родительским объектом для встроенной диаграммы является объект ChartObject, а для диаграммы на отдельном листе диаграммы родительским объектом выступает объект Workbook.

Ниже приведены операторы, которые отображают окно сообщения со словом Диаграмма.

```
MsgBox TypeName(Sheets("Лист1").ChartObjects(1).Chart)
Msgbox TypeName(Sheets("Диаграмма1"))
```

Примечание

При создании внедренной диаграммы происходит пополнение коллекций ChartObjects и Shapes, находящихся на определенном рабочем листе. (Коллекция Charts на рабочем листе не содержится.) При создании листа диаграммы пополняются коллекции Charts и Sheets, относящиеся к определенной рабочей книге.

Создание внедренной диаграммы

Объект ChartObject — это специальный тип объекта Shape. По сути, он является элементом коллекции Shapes. Для создания диаграммы воспользуйтесь методом AddChart из коллекции Shapes. Следующий оператор создает пустую внедренную диаграмму:

```
ActiveSheet.Shapes.AddChart
```

Метод AddChart использует пять аргументов (все они необязательны).

- **Type.** Этот аргумент определяет тип диаграммы. Если он не указан, используется тип диаграммы, заданный по умолчанию. Поддерживаются константы для всех типов диаграмм (например, xlArea, xlColumnClustered и т.д.).

- **Left.** Расстояние от левого края экрана до области диаграммы (выражено в пунктах). Если значение этого аргумента не указано, Excel центрирует диаграмму по горизонтали.
- **Top.** Расстояние от верхнего края экрана до области диаграммы (выражено в пунктах). Если значение этого аргумента не указано, Excel центрирует диаграмму по вертикали.
- **Width.** Ширина диаграммы (выражена в пунктах). Если значение этого аргумента не указано, Excel использует величину 354 пункта.
- **Height.** Высота диаграммы (выражено в пунктах). Если значение этого аргумента не указано, Excel использует величину 210 пунктов.

В процессе создания диаграммы часто выручает использование объектной переменной. В следующем примере создается линейчатая диаграмма, причем ссылка на нее в коде осуществляется с помощью объектной переменной `MyChart`.

```
Sub CreateChart()
   Dim MyChart As Chart
   Set MyChart = ActiveSheet.Shapes.AddChart(xlLineMarkers).Chart
End Sub
```

Диаграмма, не содержащая данных, лишена практического смысла, поэтому воспользуйтесь методом `SetSourceData` для добавления данных в только что созданную диаграмму. Следующая процедура демонстрирует применение данного метода. Она создает диаграмму, показанную на рис. 18.1.

```
Sub CreateChart()
   Dim MyChart As Chart
   Dim DataRange As Range
   Set DataRange = ActiveSheet.Range("A1:C7")
   Set MyChart = ActiveSheet.Shapes.AddChart.Chart
   MyChart.SetSourceData Source:=DataRange
End Sub
```

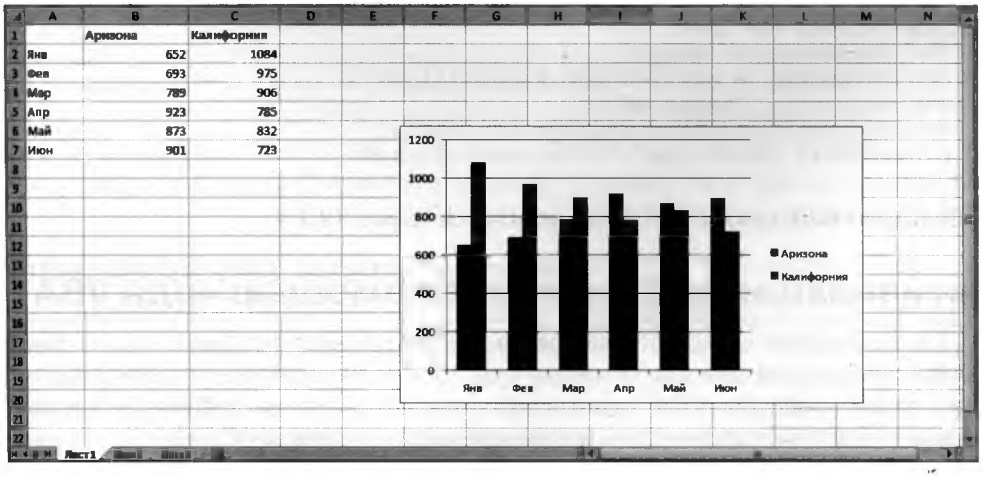


Рис. 18.1. Для создания этой диаграммы понадобилось всего лишь несколько строк кода VBA

Создание диаграммы традиционным способом

Использование метода AddChart из коллекции Shapes (как описано в разделе "Создание внедренной диаграммы") обеспечивает "новый" способ создания диаграмм, появившийся в версии Excel 2007. Исходя из соображений обратной совместимости можно использовать метод Add из коллекции ChartObjects. Этот метод, в отличие от метода AddChart из коллекции Shapes, не позволяет указывать тип диаграммы в качестве аргумента, поэтому в случае необходимости выбора типа диаграммы, отличающегося от заданного по умолчанию, следует воспользоваться свойством ChartType. Аргументы Left, Top, Width и Height также являются обязательными.

Следующая процедура использует метод Add из коллекции ChartObjects для создания внедренной диаграммы.

```
Sub CreateChart2()
   Dim MyChart As Chart
   Dim DataRange As Range
   Set DataRange = ActiveSheet.Range("A1:C7")
   Set MyChart = ActiveSheet.ChartObjects.Add(10, 10, 354, 210).Chart
   MyChart.SetSourceData Source:=DataRange
   MyChart.ChartType = xlColumnClustered
End Sub
```

Размещение диаграммы на листе диаграммы

В предыдущем разделе рассматривалась простая процедура, с помощью которой создавалась внедренная диаграмма. Если же нужно разместить диаграмму на листе диаграммы, воспользуйтесь методом Add из коллекции Charts. Этот метод применяет множество аргументов, хотя все они определяют положение листа диаграммы, а не информацию, связанную с самой диаграммой.

В следующем примере создается диаграмма на листе диаграммы, а также определяется диапазон данных и тип данных.

```
Sub CreateChartSheet()
   Dim MyChart As Chart
   Dim DataRange As Range
   Set DataRange = ActiveSheet.Range("A1:C7")
   Set MyChart = Charts.Add
   MyChart.SetSourceData Source:=DataRange
   ActiveChart.ChartType = xlColumnClustered
End Sub
```

Результат выполнения этого кода приводится на рис. 18.2.

Активизация диаграммы с помощью кода VBA

Для активизации внедренной диаграммы достаточно щелкнуть мышью в ее области. Для активизации диаграммы с помощью кода VBA воспользуйтесь методом Activate. Ниже показан оператор VBA, выполнение которого дает тот же эффект, что и щелчок мышью в области внедренной диаграммы при нажатой клавише <Ctrl>.

```
ActiveSheet.ChartObjects("Диаграмма1").Activate
```

Если диаграмма находится на листе диаграммы, используйте такой оператор:

```
Sheets("Диаграмма1").Activate
```

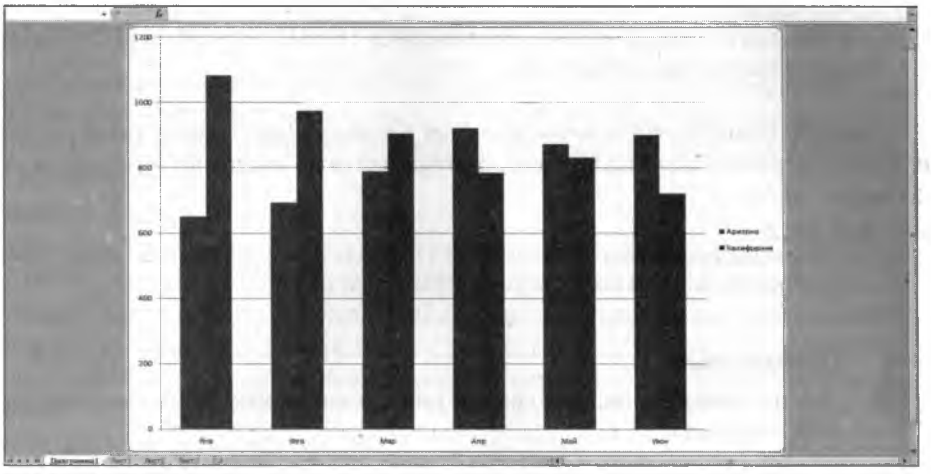


Рис. 18.2. Создание диаграммы на отдельном листе

Можно также активизировать диаграмму путем выбора содержащего ее объекта Shape.

```
ActiveSheet.Shapes("Диаграмма1").Select
```

Как только диаграмма будет активизирована, на нее можно ссылаться с помощью свойства ActiveChart. Например, выполнение следующего оператора приводит к отображению имени активной диаграммы. Если активной диаграммы в проекте нет, то оператор приведет к возникновению ошибки.

```
MsgBox ActiveChart.Name
```

Чтобы модифицировать диаграмму с помощью кода VBA, ее не обязательно активизировать. Две процедуры, представленные ниже, приводят к одинаковому результату (встроенная диаграмма Диаграмма1 любого типа превращается в диаграмму с областями). Первая процедура активизирует диаграмму перед выполнением необходимых действий, а вторая выполняет операцию без активизации.

```
Sub ModifyChart1()
   ActiveSheet.ChartObjects("Диаграмма1").Activate
   ActiveChart.ChartType = xlArea
End Sub

Sub ModifyChart2()
   ActiveSheet.ChartObjects("Диаграмма1").Chart.ChartType = xlArea
End Sub
```

Перемещение диаграммы

Внедренную диаграмму можно преобразовать в диаграмму, находящуюся на отдельном листе (лист диаграммы). Это можно сделать вручную. Просто активизируйте внедренную диаграмму и воспользуйтесь командой **Работа с диаграммами⇨Конструктор⇨Расположение⇨Переместить диаграмму** (Chart Tools⇨Design⇨Location⇨Move Chart). В диалоговом окне **Перемещение диаграммы** (Move Chart) выберите переключатель **На отдельном листе** (New Sheet) и укажите название диаграммы.

Можно также преобразовать внедренную диаграмму в лист диаграммы с помощью кода VBA. Ниже приводится пример, в котором объект ChartObject, находящийся на рабочем листе Лист1, преобразуется в лист диаграммы МояДиаграмма.

```
Sub MoveChart1()
   Sheets("Лист1").ChartObjects(1).Chart. _
   Location xlLocationAsNewSheet, "МояДиаграмма"
End Sub
```

Следующий пример прямо противоположен предыдущему — диаграмма, находящаяся на листе диаграммы МояДиаграмма, преобразуется во внедренную диаграмму на рабочем листе Лист1.

```
Sub MoveChart2()
   Charts("МояДиаграмма") _
      .Location xlLocationAsObject, "Лист1"
End Sub
```

Примечание

Для активизации перемещенной диаграммы можно воспользоваться методом Location.

Что в имени твоем?

Все объекты ChartObject имеют имена. Также именуется каждый объект Chart в составе объекта ChartObject. На первый взгляд это просто, хотя может приводить к недоразумениям. Для примера создайте диаграмму на листе Лист1 и активизируйте ее. Затем активизируйте окно отладки VBA и введите несколько команд.

```
? ActiveSheet.Shapes(1).Name
Диаграмма1
? ActiveSheet.ChartObjects(1).Name
Диаграмма1
? ActiveChart.Name
Лист1 Диаграмма1
? Activesheet.ChartObjects(1).Chart.Name
Лист1 Диаграмма1
```

Если изменить имя рабочего листа, изменится название объекта Chart, однако невозможно изменить название объекта Chart, находящегося в объекте ChartObject. Если попытаться сделать то же самое с помощью VBA, на экране отобразится сообщение об ошибке "Out of memory".

```
Activesheet.ChartObjects(1).Chart.Name = "НовоеИмя"
```

А что можно сказать относительно изменения имени объекта ChartObject? На первый взгляд, эту операцию можно выполнить в поле Имя (Name) (в левой части панели формул). К сожалению, интуиция вас подводит. Так можно переименовать форму, но диаграмму переименовать таким образом не получится (даже если диаграмма является формой). Для переименования внедренной диаграммы воспользуйтесь элементом управления Имя диаграммы (Chart Name), который находится в группе Работа с диаграммами⇨Макет⇨Свойства (Chart Tools⇨Layout⇨Properties). Этот элемент управления отображает имя активной диаграммы (фактически это название объекта ChartObject), причем этот элемент управления можно использовать для изменения названия объекта ChartObject. Как ни странно, Excel позволяет использовать имя существующего объекта ChartObject. Другими словами, на рабочем листе могут находиться десятки встроенных диаграмм, причем каждая из них будет называться Диаграмма1.

Выводы? Учитывайте эту особенность. Если обнаружится, что код VBA, предназначенный для обработки диаграмм, не работает, убедитесь в том, что диаграммы названы по-разному.

Деактивизация диаграммы

В предыдущем разделе был использован метод Activate для активизации диаграммы. Теперь же ее нужно деактивизировать, т.е. отменить выделение. Как же это сделать? В соответствии с рекомендациями справочной системы, для этого следует воспользоваться методом Deselect.

```
ActiveChart.Deselect
```

Однако этот оператор не работает, по крайней мере в данной версии Excel.

Поэтому единственный способ деактивизации диаграммы с помощью макроса — выбрать объект, отличающийся от диаграммы. В случае с внедренной диаграммой можно воспользоваться свойством RangeSelection объекта ActiveWindow для деактивизации диаграммы и выбора диапазона данных, который был выделен до активизации диаграммы.

```
ActiveWindow.RangeSelection.Select
```

Для деактивизации диаграммы, находящейся на листе диаграммы, просто напишите код VBA, который выбирает другой рабочий лист.

Определение активности диаграммы

Обычно макрос выполняет операции по отношению к активной диаграмме (выделенной пользователем). Например, макрос может изменять тип диаграммы, применять стиль или экспортировать диаграмму в графический файл.

Возникает вопрос: каким образом код VBA определяет, что пользователь выделил диаграмму? Под выделением диаграммы подразумевается активизация листа диаграммы либо щелчок мышью в области внедренной диаграммы. Первым этапом является проверка свойства TypeName для объекта Selection с помощью следующего оператора:

```
TypeName(Selection) = "Chart"
```

Фактически это выражение никогда не будет равным True. Если диаграмма активизирована, то фактически выделяется объект в составе объекта Chart. Например, выделение может представлять объект Series, объект ChartTitle, объект Legend, объект PlotArea и т.д.

Решение этой задачи заключается в том, чтобы определить, будет ли значение свойства ActiveChart равно Nothing. Если ответ на этот вопрос положителен, значит, диаграмма не является активной. В этом случае на экране отображается соответствующее сообщение, после чего процедура завершает свою работу. Код этой процедуры приведен ниже.

```
If ActiveChart Is Nothing Then
   MsgBox "Выберите диаграмму."
   Exit Sub
Else
   ' здесь находится другой код
End If
```

Иногда для определения активности диаграммы удобнее воспользоваться функцией VBA. Функция ChartIsSelected, код которой приведен ниже, возвращает значение True, если лист диаграммы активен, и значение False, если диаграмма не активизирована.

```
Private Function ChartIsSelected() As Boolean
   ChartIsSelected = Not ActiveChart Is Nothing
End Function
```

Удаление объектов из коллекции ChartObjects или Charts

Для удаления диаграммы из рабочего листа следует знать имя либо индекс объекта ChartObject. Следующий оператор удаляет объект ChartObject под именем Диаграмма1 из активного рабочего листа:

```
ActiveSheet.ChartObjects("Диаграмма1").Delete
```

Для удаления всех объектов ChartObject из рабочего листа воспользуйтесь методом Delete из коллекции ChartObjects.

```
ActiveSheet.ChartObjects.Delete
```

Для удаления внедренной диаграммы следует обратиться к коллекции Shapes. Следующий оператор удаляет диаграмму Диаграмма1 из активного рабочего листа:

```
ActiveSheet.Shapes("Диаграмма1").Delete
```

Приведенный ниже оператор удаляет все внедренные диаграммы (а также все другие формы).

```
Dim shp as Shape
For Each shp In ActiveSheet.Shapes
   shp.Delete
Next shp
```

Для удаления единственного листа диаграммы следует указать имя листа диаграммы (либо индекс). Следующий оператор удаляет лист диаграммы под названием Диаграмма1:

```
Charts("Диаграмма1").Delete
```

Для удаления всех листов диаграмм в активной рабочей книге воспользуйтесь следующим оператором:

```
ActiveWorkbook.Charts.Delete
```

При удалении всех листов диаграмм отображается предупреждающее сообщение, показанное на рис. 18.3. Для продолжения выполнения макроса пользователь должен ответить на это сообщение. Если для удаления листа используется макрос, вряд ли появление подобного сообщения вас обрадует. Если вы не хотите его видеть, воспользуйтесь следующими операторами.

```
Application.DisplayAlerts = False
ActiveWorkbook.Charts.Delete
Application.DisplayAlerts = True
```

Рис. 18.3. Это сообщение появляется при попытке удалить листы диаграмм

Циклический просмотр диаграмм

В некоторых случаях одну и ту же операцию нужно применять ко всем диаграммам. В следующем примере изменения применяются ко всем внедренным диаграммам, находящимся в активной рабочей книге. В этой процедуре осуществляется циклический об-

ход всех объектов в коллекции `ChartObjects` с последующим доступом и изменением свойств каждого объекта `Chart`.

```
Sub FormatAllCharts()
   Dim ChtObj As ChartObject
   For Each ChtObj In ActiveSheet.ChartObjects
      With ChtObj.Chart
         .ChartType = xlLineMarkers
         .ApplyLayout 3
         .ChartStyle = 12
         .ClearToMatchStyle
         .SetElement msoElementChartTitleAboveChart
         .SetElement msoElementLegendNone
         .SetElement msoElementPrimaryValueAxisTitleNone
         .SetElement msoElementPrimaryCategoryAxisTitleNone
         .Axes(xlValue).MinimumScale = 0
         .Axes(xlValue).MaximumScale = 1000
      End With
   Next ChtObj
End Sub
```

Компакт-диск

Рассматриваемый в этом разделе пример кода находится на прилагаемом к книге компакт-диске в файле `format all charts.xlsm`.

На рис. 18.4 показаны четыре диаграммы, к которым применяются различные виды форматирования. На рис. 18.5 представлены эти же диаграммы после выполнения макроса `FormatAllCharts`.

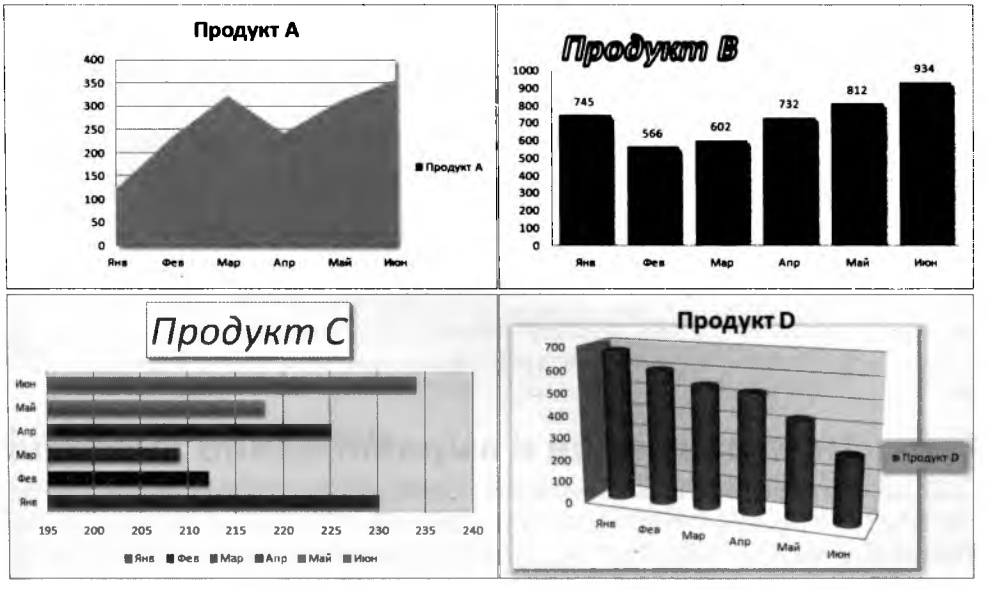


Рис. 18.4. В этих диаграммах использованы разные виды форматирования

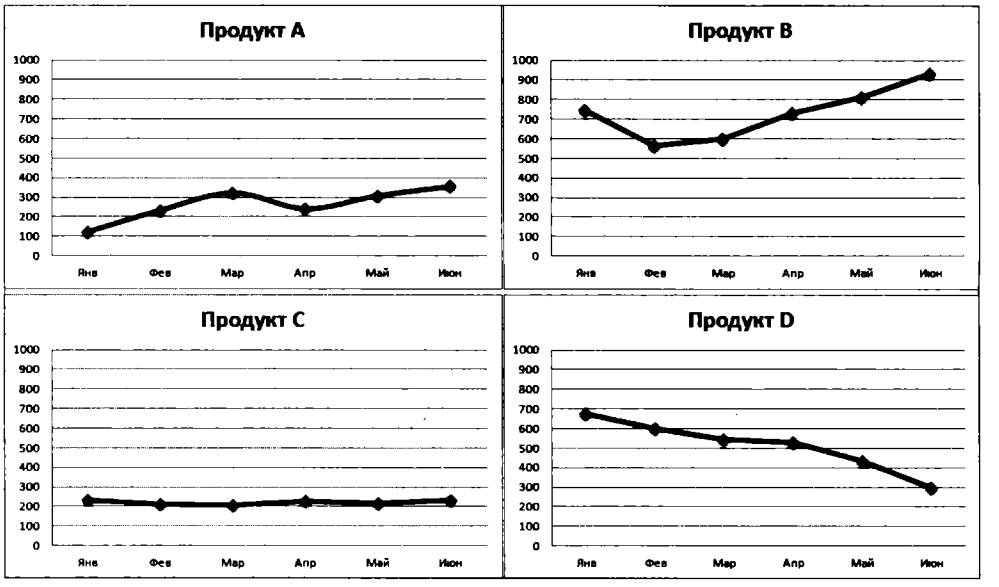


Рис. 18.5. С помощью простого макроса к четырем диаграммам было применено подходящее форматирование

Следующий макрос выполняет те же самые операции, что и процедура `FormatAllCharts`, но обрабатывает все листы диаграмм в активной рабочей книге.

```
Sub FormatAllCharts2()
   Dim cht as Chart
   For Each cht In ActiveWorkbook.Charts
      With cht
         .ChartType = xlLineMarkers
         .ApplyLayout 3
         .ChartStyle = 12
         .ClearToMatchStyle
         .SetElement msoElementChartTitleAboveChart
         .SetElement msoElementLegendNone
         .SetElement msoElementPrimaryValueAxisTitleNone
         .SetElement msoElementPrimaryCategoryAxisTitleNone
         .Axes(xlValue).MinimumScale = 0
         .Axes(xlValue).MaximumScale = 1000
      End With
   Next cht
End Sub
```

Изменение размеров и выравнивание диаграмм

Объект `ChartObject` имеет стандартные свойства, позволяющие изменять позицию (`Top` и `Left`) и размеры (`Width` и `Height`) диаграммы. Доступ к этим свойствам можно получить с помощью кода VBA. На ленте Excel также находятся элементы управления (в группе **Работа с диаграммами⇨Формат⇨Размер** (Chart Tools⇨Format⇨Size)), однако они позволяют установить лишь свойства `Height` и `Width`, но никак не `Top` и `Left`.

В следующем примере изменяются размеры всех объектов `ChartObject` на листе, в результате чего достигается соответствие с размерами активной диаграммы. Этот код также перераспределяет объекты `ChartObject` по столбцам, определенным пользователем.

```
Sub SizeAndAlignCharts()
   Dim W As Long, H As Long
   Dim TopPosition As Long, LeftPosition As Long
   Dim ChtObj As ChartObject
   Dim i As Long, NumCols As Long

   If ActiveChart Is Nothing Then
      MsgBox "Выберите образцовую диаграмму для изменения размеров"
      Exit Sub
   End If

'  Получить столбцы
   On Error Resume Next
   NumCols = InputBox("Сколько столбцов в диаграмме?")
   If Err.Number <> 0 Then Exit Sub
   If NumCols < 1 Then Exit Sub
   On Error GoTo 0

'  Получить значение размера активной диаграммы
   W = ActiveChart.Parent.Width
   H = ActiveChart.Parent.Height

'  При необходимости изменить начальные позиции
   TopPosition = 100
   LeftPosition = 20
   For i = 1 To ActiveSheet.ChartObjects.Count
      With ActiveSheet.ChartObjects(i)
         .Width = W
         .Height = H
         .Left = LeftPosition + ((i - 1) Mod NumCols) * W
         .Top = TopPosition + Int((i - 1) / NumCols) * H
      End With
   Next i
End Sub
```

Если активная диаграмма отсутствует, пользователю предлагается активизировать диаграмму, на основе которой будут изменяться размеры других диаграмм. Для ввода пользователем информации о количестве столбцов применялась функция `InputBox`. Значения свойств `Left` и `Top` вычисляются в цикле.

Компакт-диск

Рассматриваемая в этом разделе рабочая книга находится на компакт-диске в файле `size and align charts.xlsm`.

Экспорт диаграммы

Иногда диаграмму Excel нужно преобразовать в графический файл. Необходимость в этом возникает при публикации диаграммы на веб-сайте. Для этого можно воспользоваться программой для получения экранных снимков, скопировав пиксели непосредственно с экрана. Можно также написать простой макрос VBA.

В следующей процедуре метод `Export` объекта `Chart` используется для сохранения активной диаграммы в виде GIF-файла.

```
Sub SaveChartAsGIF ()
   Dim Fname as String
   If ActiveChart Is Nothing Then Exit Sub
```

```
    Fname = ThisWorkbook.Path & "\" & ActiveChart.Name & ".gif"
    ActiveChart.Export FileName:=Fname, FilterName:="GIF"
End Sub
```

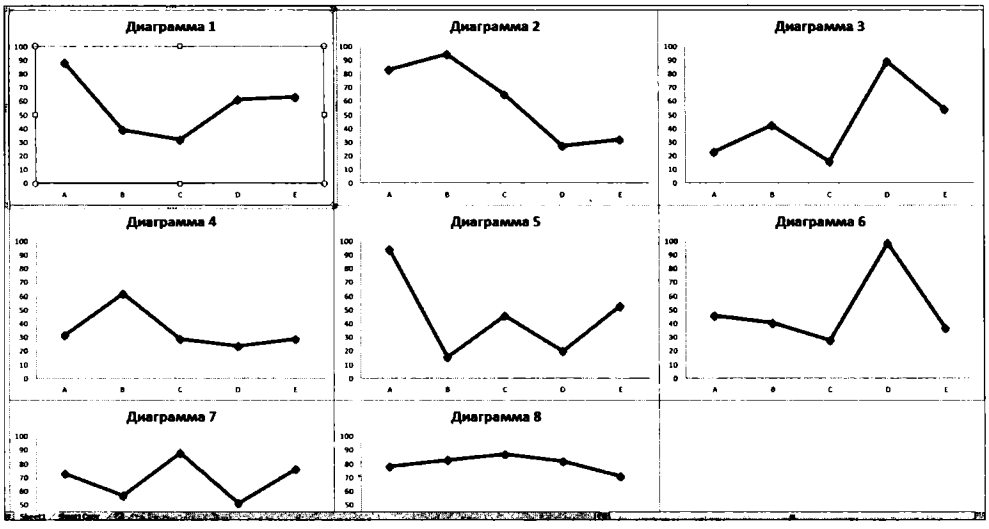


Рис. 18.6. Изменение размера и выравнивание внедренных диаграмм с помощью макроса VBA

Аргумент `FilterName` может также принимать значения `"JPEG"` и `"PNG"`. Обычно файлы GIF и PNG обеспечивают лучшее качество изображений.

Имейте в виду, что метод `Export` может оказаться неработоспособным, если на компьютере пользователя не установлен нужный фильтр экспорта изображений. Эти фильтры устанавливаются при инсталляции программ Office (или одной программы Excel).

Экспорт всех изображений

Первый способ экспорта всех изображений из рабочей книги заключается в сохранении этой книги в формате HTML. При этом создается папка, в которой находятся изображения в форматах GIF и PNG для диаграмм, форм, библиотек изображений и даже для скопированных диапазонов изображений (созданных с помощью команды Главная⇨Буфер обмена⇨Вставить⇨Вставить(А) (Home⇨Clipboard⇨Paste⇨Paste (U)).

Ниже приводится код процедуры VBA, которая автоматизирует этот процесс. Она работает с активной рабочей книгой.

```
Sub SaveAllGraphics()
    Dim FileName As String
    Dim TempName As String
    Dim DirName As String
    Dim gFile As String

    FileName = ActiveWorkbook.FullName
    TempName = ActiveWorkbook.Path & "\" & _
       ActiveWorkbook.Name & "graphics.htm"
    DirName = Left(TempName, Len(TempName) - 4) & "_файлов"

'   Сохраняет активную рабочую книгу в формате HTML,
'   затем повторно открывает исходную книгу
```

```
    ActiveWorkbook.Save
    ActiveWorkbook.SaveAs FileName:=TempName, FileFormat:=xlHtml
    Application.DisplayAlerts = False
    ActiveWorkbook.Close
    Workbooks.Open FileName

'   Удаляет HTML-файл
    Kill TempName

'   Удаляет все файлы в папке HTML, кроме файлов *.PNG
    gFile = Dir(DirName & "\*.*")
    Do While gFile <> ""
       If Right(gFile, 3) <> "png" Then Kill DirName & "\" & gFile
       gFile = Dir
    Loop
'   Показать экспортированные рисунки
    Shell "explorer.exe " & DirName, vbNormalFocus
End Sub
```

Все начинается с сохранения активной рабочей книги. Затем рабочая книга сохраняется в виде HTML-файла и файл закрывается, после чего повторно открывается исходная рабочая книга. После этого HTML-файл удаляется, поскольку нас больше интересует содержимое созданной папки (именно здесь находятся изображения). Затем осуществляется просмотр папки, в ходе которого удаляются все файлы, за исключением PNG. И наконец, используется функция `Shell` для отображения содержимого этой папки.

Компакт-диск

Рассматриваемый в этом разделе пример находится на прилагаемом к книге компакт-диске в файле `export all graphics.xlsm`.

Изменение применяемых в диаграмме данных

В рассмотренных ранее примерах использовалось свойство `SourceData`, определяющее полный диапазон данных для диаграммы. Часто необходимо изменить данные, применяемые в том или ином ряде данных. Для этого используется свойство `Values` объекта `Series`. Объект `Series` также включает свойство `XValues`, используемое для хранения значений оси категорий.

Примечание

Свойство `Values` соответствует третьему аргументу функции `РЯД`, а свойство `XValues` — второму аргументу этой же функции (см. врезку "Функция `РЯД` в диаграмме").

Функция РЯД в диаграмме

Данные, используемые в каждом ряде (последовательности) данных диаграммы, определяются с помощью функции `РЯД` (SERIES). После выделения ряда данных в диаграмме на панели формул отображается функция `РЯД`. Это не "настоящая" функция, ее нельзя использовать в ячейке рабочего листа; кроме того, в ней нельзя применять другие функции. Однако допускается редактирование аргументов функции `РЯД`.

Функция `РЯД` имеет следующий синтаксис.

```
=РЯД(имя_ряда, подписи_категорий, значения, порядок, размеры)
```

Ниже описаны аргументы функции РЯД.

- *Имя_ряда* (необязательно). Ссылка на ячейку, которая содержит название ряда, применяемое в легенде. Если диаграмма содержит только один ряд данных, этот аргумент используется в качестве заголовка. В качестве значения этого аргумента может также использоваться текст в кавычках. Если этот аргумент пропущен, Excel создает имя ряда, заданное по умолчанию (например, Ряд1).
- *Подписи_категорий* (обязательно). Ссылка на диапазон данных, которая содержит подписи оси категорий. Если этот аргумент пропущен, Excel использует последовательные целые числа, которые начинаются с 1. Для графиков этот аргумент определяет значения по оси абсцисс. Допускаются также ссылки на несмежные диапазоны. Адреса диапазонов разделяются запятыми и заключаются в скобки. В качестве значения этого аргумента может также использоваться массив, включающий разделенные запятыми значения (либо текст в кавычках) и заключенный в фигурные скобки.
- *Значения* (обязательно). Ссылка на диапазон данных, которая содержит значения ряда данных. Для графиков этот аргумент указывает значения по оси ординат. Допускаются также ссылки на несмежные диапазоны. Адреса диапазонов разделяются запятыми и заключаются в скобки. Значение аргумента может также представлять собой массив разделенных запятыми значений, заключенных в фигурные скобки.
- *Порядок* (обязательно). Целое число, которое указывает порядок представления рядов данных. Этот аргумент используется тогда, когда диаграмма включает несколько рядов данных. Например, в гистограмме с накоплением этот параметр определяет порядок формирования стека. Не допускается использование ссылки на ячейку.
- *Размеры* (только для пузырьковых диаграмм). Ссылка на диапазон данных, который содержит значения размеров пузырьков в пузырьковой диаграмме. Также допускаются ссылки на несмежные диапазоны. Адреса диапазонов разделяются запятыми и заключаются в скобки. В качестве значения аргумента может также использоваться массив значений, заключенных в фигурные скобки.

Ссылки на диапазоны в функции РЯД всегда абсолютные и включают имя листа.

```
=РЯД(Лист1!$B$1,,Лист1!$B$2:$B$7,1)
```

Ссылка может задаваться на несмежный диапазон. Если это так, то каждый поддиапазон разделяется точкой с запятой, а аргумент заключается в скобки. В следующей формуле в качестве значений используются диапазоны B2:B3 и B5:B7.

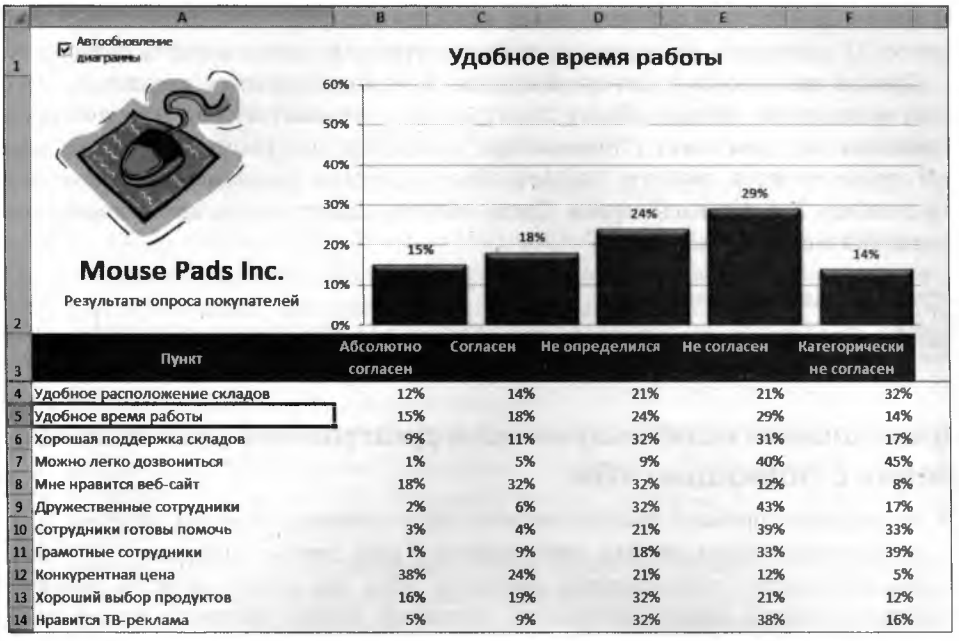
```
=РЯД(,,(Лист1!$B$2:$B$3,Лист1!$B$5:$B$7),1)
```

Вместо ссылок на диапазоны можно вставить их имена. Если поступать подобным образом (и если имя задается на уровне рабочей книги), Excel изменит ссылку в функции РЯД таким образом, чтобы включить рабочую книгу.

```
=РЯД(Лист1!$B$1,,budget.xlsx!CurrentData,1)
```

Изменение данных диаграммы на основе активной ячейки

На рис. 18.7 показана диаграмма, построенная на основе данных строки, в которой находится активная ячейка. Как только пользователь перемещает курсор в ячейке, диаграмма автоматически обновляется.



Пункт	Абсолютно согласен	Согласен	Не определился	Не согласен	Категорически не согласен
Удобное расположение складов	12%	14%	21%	21%	32%
Удобное время работы	15%	18%	24%	29%	14%
Хорошая поддержка складов	9%	11%	32%	31%	17%
Можно легко дозвониться	1%	5%	9%	40%	45%
Мне нравится веб-сайт	18%	32%	32%	12%	8%
Дружественные сотрудники	2%	6%	32%	43%	17%
Сотрудники готовы помочь	3%	4%	21%	39%	33%
Грамотные сотрудники	1%	9%	18%	33%	39%
Конкурентная цена	38%	24%	21%	12%	5%
Хороший выбор продуктов	16%	19%	32%	21%	12%
Нравится ТВ-реклама	5%	9%	32%	38%	16%

Рис. 18.7. Эта диаграмма всегда отображает данные из строки, в которой находится активная ячейка

В этом примере используется обработчик событий объекта `Лист1`. Событие `SelectionChange` происходит в случае, когда пользователь изменяет выделенную область путем перемещения курсора ячейки. Ниже приводится код процедуры обработки событий (находится в модуле кода для объекта `Лист1`).

```
Private Sub Worksheet_SelectionChange(ByVal Target _
  As Excel.Range)
    If CheckBox1 Then Call UpdateChart
End Sub
```

Другими словами, при каждом перемещении курсора ячейки вызывается на выполнение процедура `Worksheet_SelectionChange`. Если установлен флажок **Автообновление диаграммы** — элемент управления ActiveX на листе, — вызывается процедура `UpdateChart`, код которой приведен ниже.

```
Sub UpdateChart()
  Dim ChtObj As ChartObject
  Dim UserRow As Long
  Set ChtObj = ActiveSheet.ChartObjects(1)
  UserRow = ActiveCell.Row
  If UserRow < 4 Or IsEmpty(Cells(UserRow, 1)) Then
    ChtObj.Visible = False
  Else
    ChtObj.Chart.SeriesCollection(1).Values = _
      Range(Cells(UserRow, 2), Cells(UserRow, 6))
    ChtObj.Chart.ChartTitle.Text = Cells(UserRow, 1).Text
    ChtObj.Visible = True
  End If
End Sub
```

Переменная UserRow содержит номер строки, в которой находится активная ячейка. Оператор If проверяет, находится ли активная ячейка в строке, которая содержит данные. (Данные начинаются с четвертой строки.) Если курсор ячейки находится в строке, которая не содержит данных, объект ChartObject скрывается, и на экране появляется находящийся под ним текст ("Невозможно отобразить диаграмму"). Если же данные в этой строке имеются, свойству Values объекта Series присваивается диапазон данных в столбцах 2–6 активной строки. Также объекту ChartTitle присваивается текст, находящийся в столбце A.

Компакт-диск

Пример из этого раздела находится на прилагаемом к книге компакт-диске в файле chart active cell.xlsm.

Определение используемых в диаграмме диапазонов данных с помощью VBA

В предыдущем примере рассматривалось использование свойства Values объекта Series для определения данных, применяемых в ряде данных диаграммы. В этом разделе рассматривается использование макросов VBA для идентификации диапазонов, применяемых рядами данных диаграммы. Например, можно увеличить размер каждого ряда данных путем добавления новой ячейки в диапазон.

Ниже описаны три свойства, используемые в процессе решения этой задачи.

- **Свойство Formula.** Возвращает значение функции РЯД либо настраивает ее с учетом выбранного ряда. После выбора ряда данных на диаграмме соответствующая функция РЯД отображается на панели формул. Свойство Formula возвращает функцию РЯД в виде строки.
- **Свойство Values.** Возвращает или устанавливает коллекцию всех значений ряда. Может быть диапазоном данных рабочего листа либо массивом постоянных значений, но не может быть тем и другим одновременно.
- **Свойство XValues.** Возвращает или устанавливает массив X-значений для ряда данных диаграммы. Свойству XValues можно присвоить диапазон данных на рабочем листе или массив значений, но не комбинацию диапазона данных и массива значений. Свойство XValues может быть также пустым.

При создании макроса VBA, для которого требуется обозначить диапазон данных, используемый для определения ряда данных, можно рассматривать свойство Values объекта Series в качестве "билета". Создается впечатление, что свойство Xvalues обеспечивает способ получения диапазона данных, который содержит X-значения (или надписи на оси категорий). На первый взгляд это представление *кажется* верным (в теории), но на практике оно не работает.

При установке значения свойства Values для объекта Series можно указать объект Range либо массив. Учтите, что при чтении значения этого свойства всегда возвращается массив. К сожалению, объектная модель не может подсказать способа получения доступа к объекту Range с помощью объекта Series.

Одно из возможных решений заключается в том, чтобы написать код, производящий синтаксический анализ функции РЯД, в процессе которого извлекается диапазон адресов.

На первый взгляд эта задача кажется простой, но на практике функция РЯД может быть очень сложной. Ниже приведены примеры этой функции.

```
=РЯД(Лист1!$B$1,Лист1!$A$2:$A$4,Лист1!$B$2:$B$4,1)
=РЯД(,,Лист1!$B$2:$B$4,1)
=РЯД(,Лист1!$A$2:$A$4,Лист1!$B$2:$B$4,1)
=РЯД("Итоги продаж",,Лист1!$B$2:$B$4,1)
=РЯД(,{"Январь","Февраль","Март"},Лист1!$B$2:$B$4,1)
=РЯД(,(Лист1!$A$2,Лист1!$A$4),(Лист1!$B$2,Лист1!$B$4),1)
=РЯД(Лист1!$B$1,Лист1!$A$2:$A$4,Лист1!$B$2:$B$4,1,Лист1!$C$2:$C$4)
```

Таким образом, в функции РЯД аргументы могут отсутствовать, в качестве аргументов могут использоваться массивы и даже несмежные диапазоны адресов. И, что еще больше усложняет дело, пузырьковая диаграмма в последнем примере функции РЯД может включать дополнительный аргумент. Синтаксический разбор подобного аргумента является нетривиальной программистской задачей.

Я провел много времени, решая эту задачу, и в итоге пришел к верному решению. Суть его заключалась в обращении к функции-"пустышке". Эта функция принимает те же аргументы, что и функция РЯД, возвращая массив размером 2×5, в котором содержатся все сведения о функции РЯД.

Полученное решение было упрощено путем создания четырех пользовательских функций VBA, каждая из которых принимает один аргумент (ссылка на объект `Series`) и возвращает массив, состоящий из двух элементов. Эти функции перечислены ниже.

- `SERIESNAME_FROM_SERIES`. Первый элемент массива, содержащий строку, которая описывает тип данных для первого аргумента функции РЯД (диапазон, пусто или строка). Второй элемент массива содержит адрес диапазона, пустую строку или строку.
- `XVALUES_FROM_SERIES`. Первый элемент массива содержит строку, которая описывает тип данных второго аргумента функции РЯД (диапазон, массив, пусто или строка). Второй элемент массива содержит адреса диапазона, массив, пустую строку или строку.
- `VALUES_FROM_SERIES`. Первый элемент массива содержит строку, которая описывает тип данных третьего аргумента функции РЯД (диапазон или массив). Второй элемент массива содержит адрес диапазона или массива.
- `BUBBLESIZE_FROM_SERIES`. Первый элемент массива содержит строку, которая описывает тип данных пятого аргумента функции РЯД (диапазон, массив или пусто). Второй элемент массива содержит адрес диапазона, массив или пустую строку. Эта функция работает только с пузырьковыми диаграммами.

Обратите внимание на, что я обошел вниманием функцию, используемую для получения четвертого аргумента функции РЯД (порядок рисования). Этот аргумент можно получить непосредственно с помощью свойства `PlotOrder` объекта `Series`.

Компакт-диск

Код VBA, реализующий описанные выше функции, слишком длинный и здесь не приводится, но вы можете его найти на прилагаемом к книге компакт-диске в файле `get series ranges.xlsm`. Эти функции документированы и могут быть легко адаптированы для применения в других ситуациях.

В следующем примере демонстрируется применение функции VALUES_FROM_SERIES. Она отображает адрес диапазона для первого ряда данных в активной диаграмме.

```
Sub ShowValueRange()
   Dim Ser As Series
   Dim x As Variant
   Set Ser = ActiveChart.SeriesCollection(1)
   x = VALUES_FROM_SERIES(Ser)
   If x(1) = "Range" Then
      MsgBox Range(x(2)).Address
   End If
End Sub
```

Переменная x определена как вариант и может хранить двухэлементный массив, который возвращается функцией VALUES_FROM_SERIES. Первый элемент массива X-значений содержит строку, которая описывает тип данных. Если строка представляет собой диапазон значений, в окне сообщения отображается адрес диапазона, содержащегося во втором элементе массива X-значений.

На рис. 18.8 показан еще один пример. Эта диаграмма включает три ряда данных. Кнопки, находящиеся на рабочем листе, вызывают макрос, который выполняет расширение и сжатие каждого диапазона данных.

Рис. 18.8. На примере этой рабочей книги демонстрируются способы расширения и сжатия ряда данных диаграммы с помощью макроса VBA

Ниже приводится код процедуры ContractAllSeries. Эта процедура выполняет циклический обход коллекции SeriesCollection, а также использует функции XVALUE_FROM_SERIES и VALUES_FROM_SERIES для выборки текущих диапазонов данных. Затем применяется метод Resize для уменьшения размера диапазонов данных.

```
Sub ContractAllSeries()
   Dim s As Series
   Dim Result As Variant
   Dim DRange As Range
   For Each s In _
      ActiveSheet.ChartObjects(1).Chart.SeriesCollection
      Result = XVALUES_FROM_SERIES(s)
      If Result(1) = "Диапазон" Then
```

```
            Set DRange = Range(Result(2))
            If DRange.Rows.Count > 1 Then
               Set DRange = DRange.Resize(DRange.Rows.Count - 1)
               s.XValues = Drange
            End If
         End If
         Result = VALUES_FROM_SERIES(s)
         If Result(1) = "Диапазон" Then
            Set DRange = Range(Result(2))
            If DRange.Rows.Count > 1 Then
               Set DRange = DRange.Resize(DRange.Rows.Count - 1)
               s.Values = Drange
            End If
         End If
   Next s

End Sub
```

Процедура `ExpandAllSeries` подобна описанной выше процедуре. После ее вызова на выполнение каждый диапазон данных расширяется на одну ячейку.

Отображение подписей для данных на диаграмме

Чаще всего пользователи Excel жалуются на несовершенство системы добавления подписей на диаграммы. В качестве примера рассмотрим точечную диаграмму на рис. 18.9. Было бы неплохо отобразить подписи для каждой точки данных представленного диапазона, хотя можно провести в поисках весь день, но так и не найти команды Excel, которая позволит разместить необходимые подписи в области диаграммы (такой команды просто не существует). Подписи к точкам данных ограничены только реальными значениями, если, конечно, не отредактировать каждую точку вручную и не ввести самостоятельно требуемый текст.

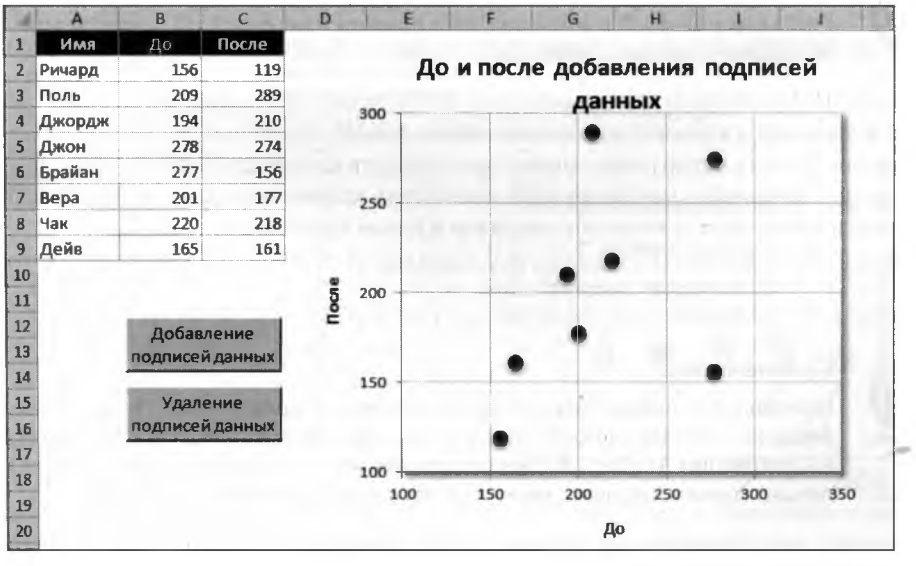


Рис. 18.9. Точечная диаграмма без подписей данных

Процедура DataLabelsFromRange работает только с первой диаграммой на активном листе. При выполнении этой процедуры пользователь должен указать диапазон данных, после чего осуществляется циклический обход коллекции Points с изменением свойства Text для значений, находящихся в диапазоне данных.

```
Sub DataLabelsFromRange()
   Dim DLRange As Range
   Dim Cht As Chart
   Dim i As Integer, Pts As Integer

'  Определение диаграммы
   Set Cht = ActiveSheet.ChartObjects(1).Chart

'  Запрос диапазона
   On Error Resume Next
   Set DLRange = Application.InputBox _
      (prompt:="Укажите диапазон с подписями", Type:=8)
   If DLRange Is Nothing Then Exit Sub
   On Error GoTo 0

'  Добавление подписей
   Cht.SeriesCollection(1).ApplyDataLabels _
      Type:=xlDataLabelsShowValue, _
      AutoText:=True, _
      LegendKey:=False

'  Просмотр диапазона и назначение подписей
   Pts = Cht.SeriesCollection(1).Points.Count
   For i = 1 To Pts
      Cht.SeriesCollection(1). _
      Points(i).DataLabel.Text = DLRange(i)
   Next i
End Sub
```

Компакт-диск

Рассматриваемый в этом разделе пример находится на прилагаемом к книге компакт-диске в файле data labels.xlsm.

На рис. 18.10 показана диаграмма после выполнения процедуры DataLabelsFromRange и указания в качестве диапазона данных A2:A9.

Подпись данных в диаграмме может представлять собой ссылку на ячейку. Для изменения кода процедуры DataLabelsFromRange таким образом, чтобы создавались ссылки на ячейки, просто измените оператор в цикле For-Next.

```
Cht.SeriesCollection(1).Points(i).DataLabel.Text = _
   "=" & "'" & DLRange.Parent.Name & "'!" & _
   DLRange(i).Address(ReferenceStyle:=xlR1C1)
```

Примечание

Описанная выше процедура несовершенна и практически не выполняет проверку на наличие ошибок. Кроме того, она работает только с первым объектом из коллекции Series. Более совершенную процедуру, выполняющую добавление подписей данных, можно найти в надстройке Power Utility Pak.

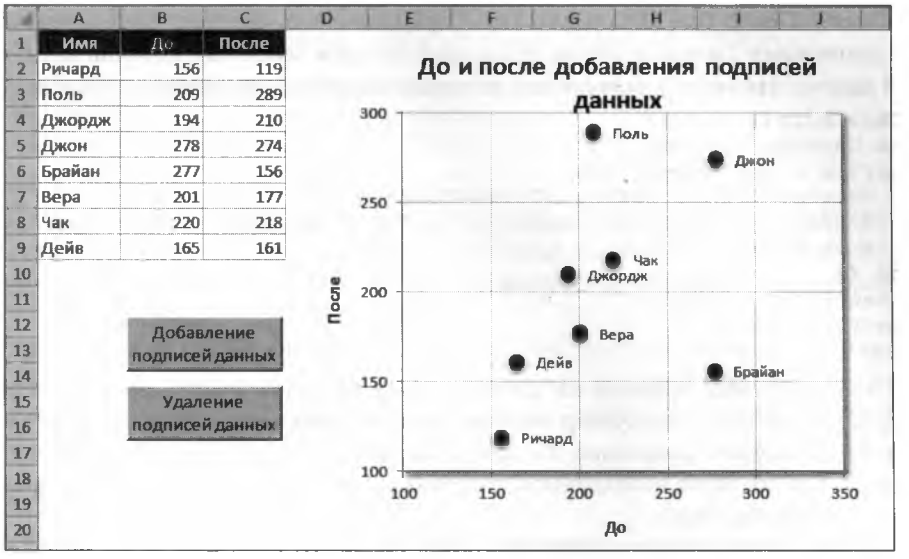


Рис. 18.10. Точечная диаграмма приобрела новый вид благодаря процедуре VBA

Отображение диаграммы в пользовательском диалоговом окне

В главе 15 рассмотрен способ отображения диаграммы в пользовательском диалоговом окне (`UserForm`). Описанный метод заключается в сохранении диаграммы в виде файла формата GIF, после чего необходимо загрузить последний в элемент управления `Image`, размещаемый в диалоговом окне `UserForm`.

В рассматриваемом примере используется аналогичная методика, но добавляется еще кое-что: диаграмма, созданная "на лету", которая использует данные в строке с активной ячейкой. Этот пример показан на рис. 18.11.

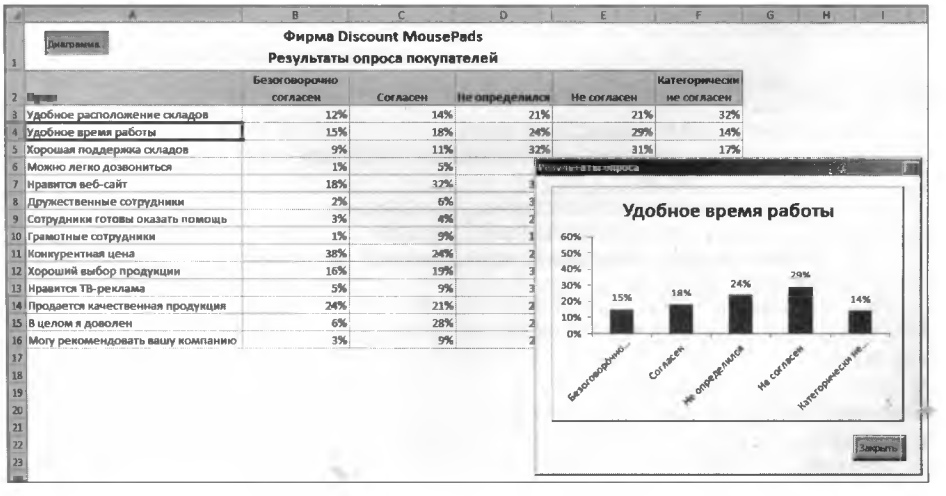


Рис. 18.11. Диаграмма, отображаемая в пользовательском диалоговом окне, создана "на лету" на основе данных в активной строке

Диалоговое окно UserForm для этого примера очень простое. Оно содержит элементы управления Image и CommandButton (кнопка Закрыть). Рабочий лист, содержащий данные, также включает кнопку, которая вызывает следующую процедуру.

```
Sub ShowChart()
   Dim UserRow As Long
   UserRow = ActiveCell.Row
   If UserRow < 2 Or IsEmpty(Cells(UserRow, 1)) Then
      MsgBox "Переместите курсор к ячейке с данными диапазона."
      Exit Sub
   End If
   CreateChart (UserRow)
   UserForm1.Show
End Sub
```

Так как диаграмма основана на данных строки активной ячейки, процедура отображает предупреждение, если курсор оказывается в неправильной строке данных. Если активная строка выбрана правильно, то процедура ShowChart вызывает процедуру CreateChart для создания необходимой диаграммы. После этого отображается пользовательское диалоговое окно.

Процедура CreateChart принимает один аргумент, который представляет строку активной ячейки. Эта процедура получена на основе макроса, который записан при создании диаграммы и откорректирован для приведения к более универсальному виду.

```
Sub CreateChart(r)
   Dim TempChart As Chart
   Dim CatTitles As Range
   Dim SrcRange As Range, SourceData As Range
   Dim FName As String

   Set CatTitles = ActiveSheet.Range("A2:F2")
   Set SrcRange = ActiveSheet.Range(Cells(r, 1), Cells(r, 6))
   Set SourceData = Union(CatTitles, SrcRange)

'  Добавление диаграммы
   Application.ScreenUpdating = False
   Set TempChart = ActiveSheet.Shapes.AddChart.Chart
   TempChart.SetSourceData Source:=SourceData

'  Настройка диаграммы
   With TempChart
      .ChartType = xlColumnClustered
      .SetSourceData Source:=SourceData, PlotBy:=xlRows
      .HasLegend = False
      .PlotArea.Interior.ColorIndex = xlNone
      .Axes(xlValue).MajorGridlines.Delete
      .ApplyDataLabels Type:=xlDataLabelsShowValue, _
         LegendKey:=False
      .Axes(xlValue).MaximumScale = 0.6
      .ChartArea.Format.Line.Visible = False
   End With
'  Задание размера диаграммы
   With ActiveSheet.ChartObjects(1)
      .Width = 300
      .Height = 200
   End With
```

```
' Сохранение диаграммы в формате GIF
   FName = ThisWorkbook.Path & Application.PathSeparator & _
      temp.gif"
   TempChart.Export Filename:=FName, filterName:="GIF"
   ActiveSheet.ChartObjects(1).Delete
   Application.ScreenUpdating = True

End Sub
```

После завершения работы процедуры `CreateChart` рабочий лист будет содержать объект `ChartObject` с диаграммой данных на основе строки активной ячейки. Но объект `ChartObject` останется невидимым, так как свойство `ScreenUpdating` объекта `Application` отключено.

Последний оператор в процедуре `ShowChart` загружает диалоговое окно `UserForm`. Ниже приведен листинг процедуры `UserForm_Initialize`. Эта процедура сохраняет диаграмму в виде файла GIF, удаляет объект `ChartObject` и загружает файл формата GIF в элемент управления `Image`.

```
Private Sub UserForm_Initialize()
   Dim FName As String
   FName = ThisWorkbook.Path & _
      Application.PathSeparator & "temp.gif"
   UserForm1.Image1.Picture = LoadPicture(FName)
End Sub
```

Компакт-диск

Рассматриваемая в этом разделе рабочая книга находится на прилагаемом компакт-диске в файле `chart in userform.xlsm`.

События диаграмм

В Excel поддерживается ряд событий, связанных с диаграммами. Например, активизация диаграммы приводит к генерированию события `Activate`. Событие `Calculate` происходит в случае, когда диаграмма принимает новые либо измененные данные. Можно также написать код VBA, который вызывается, когда происходит определенное событие.

Перекрестная ссылка

Для получения дополнительных сведений о событиях обратитесь к главе 19.

В табл. 18.1 перечислены все события диаграмм.

Таблица 18.1. События объекта `Chart`

Событие	Действие, приводящее к возникновению события
`Activate`	Активизирован лист диаграммы или внедренная диаграмма
`BeforeDoubleClick`	На встроенной диаграмме выполнен двойной щелчок. Это событие происходит перед принятой по умолчанию реакцией на двойной щелчок мышью
`BeforeRightClick`	На встроенной диаграмме выполнен щелчок правой кнопкой мыши. Это событие происходит перед принятой по умолчанию реакцией на щелчок правой кнопкой мыши
`Calculate`	На диаграмме отображаются новые или обновленные данные

Окончание табл. 18.1

Событие	Действие, приводящее к возникновению события
`Deactivate`	Диаграмма деактивизируется
`MouseDown`	Кнопка мыши нажата, а указатель находится над диаграммой
`MouseMove`	Расположение указателя мыши изменяется, пока он находится над диаграммой
`MouseUp`	Кнопка мыши отпущена, пока указатель находится над диаграммой
`Resize`	Изменение размера диаграммы
`Select`	Выделение одного из элементов диаграммы
`SeriesChange`	Изменение значения точки данных на диаграмме

Пример использования событий объекта `Chart`

Для того чтобы создать процедуру обработки события, которое происходит на листе диаграммы, код VBA должен быть сохранен в модуле кода объекта `Chart`. Чтобы активизировать этот модуль кода, дважды щелкните на элементе `Chart` в окне проектов. После этого в модуле кода выберите `Chart` из раскрывающегося списка объектов в левой части окна и укажите событие в раскрывающемся списке `Procedure`, находящемся в правой части экрана (рис. 18.12).

Примечание

Поскольку для встроенной диаграммы модуль кода не представлен, описанная в этом разделе процедура применяется только к листам диаграмм. Конечно, события можно обрабатывать и для встроенных диаграмм, но для этого нужно выполнить кропотливую работу, связанную с созданием модуля класса. Эта процедура описана в разделе "Поддержка событий для встроенных диаграмм".

Рис. 18.12. Выбор события в модуле кода для объекта `Chart`

Следующий код отображает сообщение, когда пользователь активизирует лист диаграммы, деактивизирует его либо выбирает любой элемент диаграммы. Я создал рабочую книгу с листом диаграммы, а также написал три процедуры обработки событий, перечисленные ниже.

- `Chart_Activate`. Вызывается после активизации листа диаграммы.
- `Chart_Deactivate`. Вызывается после деактивизации листа диаграммы.
- `Chart_Select`. Вызывается после выбора элемента листа диаграммы.

Компакт-диск

Рассматриваемую в этом разделе рабочую книгу можно найти на прилагаемом компакт-диске в файле `events - chart sheet.xlsm`.

Ниже приводится код процедуры `Chart_Activate`.

```
Private Sub Chart_Activate()
   Dim msg As String
   msg = "Привет, " & Application.UserName & vbCrLf & vbCrLf
   msg = msg & "Результаты продаж за 6 месяцев "
   msg = msg & "итого по продуктам 1-3." & vbCrLf & vbCrLf
   msg = msg & _
      "Щелкните на элементе диаграммы, чтобы выяснить его природу."
   MsgBox msg, vbInformation, ActiveWorkbook.Name
End Sub
```

Эта процедура отображает сообщение всякий раз, когда диаграмма активизируется (рис. 18.13).

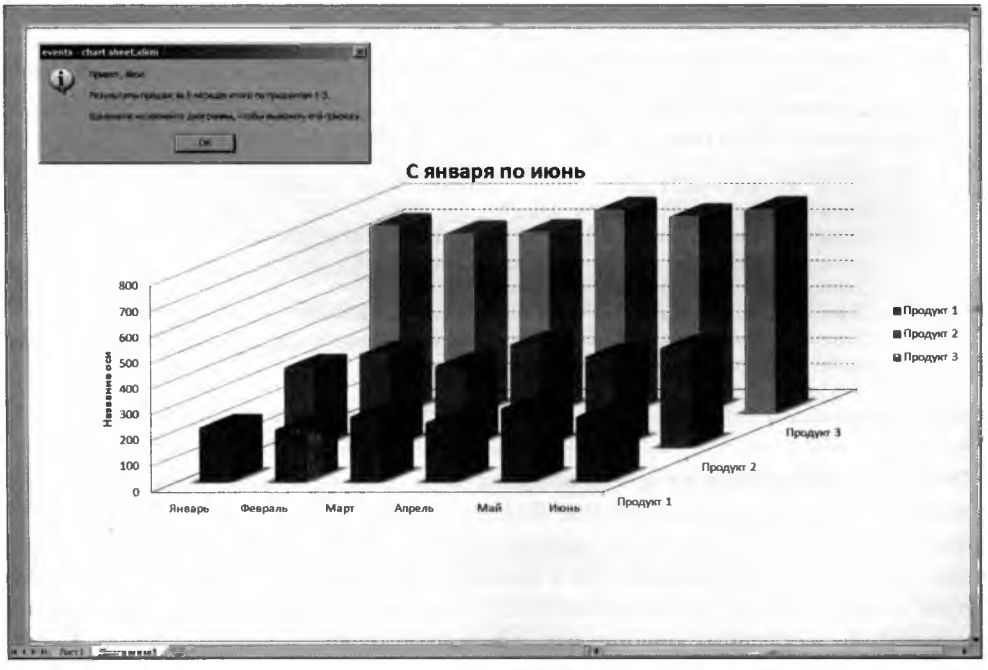


Рис. 18.13. В случае активизации диаграммы вызывается событие `Chart_Activate`, которое приводит к появлению соответствующего сообщения

Процедура `Chart_Deactivate` также отображает сообщение, но только в том случае, когда лист диаграммы деактивизируется.

```
Private Sub Chart_Deactivate()
   Dim msg As String
```

```
   msg = "Спасибо за просмотр диаграммы."
   MsgBox msg, , ActiveWorkbook.Name
End Sub
```

Процедура `Chart_Select`, код которой приведен ниже, вызывается после выбора произвольного элемента диаграммы.

```
Private Sub Chart_Select(ByVal ElementID As Long, _
   ByVal Arg1 As Long, ByVal Arg2 As Long)
      Dim Id As String
      Select Case ElementID
         Case xlAxis: Id = "Ось"
         Case xlAxisTitle: Id = "Название оси"
         Case xlChartArea: Id = "Область диаграммы"
         Case xlChartTitle: Id = "Заголовок диаграммы"
         Case xlCorners: Id = "Углы"
         Case xlDataLabel: Id = "Подпись данных"
         Case xlDataTable: Id = "Таблица данных"
         Case xlDownBars: Id = "Нижние полосы"
         Case xlDropLines: Id = "Подзаголовок"
         Case xlErrorBars: Id = "Погрешность"
         Case xlFloor: Id = "Основание"
         Case xlHiLoLines: Id = "Верхние/нижние линии"
         Case xlLegend: Id = "Легенда"
         Case xlLegendEntry: Id = "Запись легенды"
         Case xlLegendKey: Id = "Ключ легенды"
         Case xlMajorGridlines: Id = "Базовые линии сетки"
         Case xlMinorGridlines: Id = "Вспомогательные линии сетки"
         Case xlNothing: Id = "Ничего"
         Case xlPlotArea: Id = "Область построения"
         Case xlRadarAxisLabels: Id = "Подписи оси лепестка"
         Case xlSeries: Id = "Ряд данных"
         Case xlSeriesLines: Id = "Линии ряда данных"
         Case xlShape: Id = "Форма"
         Case xlTrendline: Id = "Тенденция"
         Case xlUpBars: Id = "Верхние полосы"
         Case xlWalls: Id = "Стенки"
         Case xlXErrorBars: Id = "Горизонтальная погрешность"
         Case xlYErrorBars: Id = "Вертикальная погрешность"
         Case Else:: Id = "Нечто неизвестное"
      End Select
MsgBox "Тип выделения:" & Id & vbCrLf & Arg1 & vbCrLf & Arg2
End Sub
```

Эта процедура отображает окно сообщения, в котором находится описание выделенного элемента. Если произойдет событие `Select`, аргумент `ElementID` будет содержать целое число, которое соответствует выделенному элементу. Аргументы `Arg1` и `Arg2` содержат дополнительную информацию о выделенном элементе (дополнительные сведения об этом можно найти в справочной системе). Структура `Select Case` применяется для преобразования встроенных констант в описательные строки.

Примечание

Это далеко не полный список элементов, входящих в состав объекта `Chart`. Именно поэтому пришлось воспользоваться оператором `Case Else`.

Поддержка событий для встроенных диаграмм

Как отмечалось в предыдущем разделе, события объекта `Chart` автоматически реализованы для листов диаграмм, но отключены для диаграмм, встроенных в рабочий лист. Для того чтобы использовать события, связанные со встроенной диаграммой, необходимо выполнить следующие действия.

Создайте модуль класса

В VBE выберите проект в окне Project (Проект) и выполните команду Insert⇨Class Module (Вставить⇨Модуль класса). Это приведет к добавлению пустого модуля класса в проект. Если возникнет желание, можно назначить модулю класса более описательное имя (например, `clsChart`) в окне Properties (Свойства). Переименовывать модуль класса не обязательно, хотя часто полезно.

Объявите глобальный объект `Chart`

Следующим шагом является создание глобальной переменной, которая используется в качестве имени класса. Переменная должна иметь тип `Chart` и объявляться в модуле класса с ключевым словом `WithEvents`. Если опустить ключевое слово `WithEvents`, то объект не будет реагировать на события. Ниже приведен пример соответствующего объявления.

```
Public WithEvents clsChart As Chart
```

Свяжите объявленный объект с диаграммой

Перед тем как будут запущены процедуры обработки событий, необходимо связать объявленный объект в модуле класса со встроенной диаграммой. Для этого следует объявить объект типа `clsChart` (в данном случае используется имя модуля класса). Это должна быть переменная уровня модуля, которая объявлена в обычном модуле VBA (а не в модуле класса). Ниже приводится пример.

```
Dim MyChart As New clsChart
```

Введите код, который фактически создаст экземпляр объекта `clsChart`. Воспользуйтесь представленным ниже оператором.

```
Set MyChart.clsChart = ActiveSheet.ChartObjects(1).Chart
```

После выполнения предыдущего оператора объект `clsChart` в модуле класса будет указывать на первую встроенную диаграмму активного листа. Следовательно, процедуры обработки событий в модуле класса будут выполняться при возникновении соответствующих событий.

Создайте процедуры обработки событий для класса диаграммы

В этом разделе речь пойдет о том, как создавать процедуры обработки событий в модуле класса. Помните, что модуль класса должен содержать следующее объявление.

```
Public WithEvents clsChart As Chart
```

После объявления с помощью ключевого слова `WithEvents` новый объект появится в раскрывающемся списке Object (Объект) в модуле класса. Если вы выберете новый объект в поле Object, его действительные события будут перечислены в раскрывающемся списке Procedure (Процедура) в правой части окна.

Приведенный далее пример является простой процедурой обработки события, которая выполняется при активизации встроенной диаграммы. Эта процедура создает окно

сообщения, которое содержит имя "родителя" объекта `Chart` (в его роли выступает объект `ChartObject`).

```
Private Sub clsChart_Activate()
   MsgBox clsChart.Parent.Name & " активизирован!"
End Sub
```

Компакт-диск

На прилагаемом компакт-диске находится рабочая книга (файл под названием `events - embedded chart.xlsm`), которая демонстрирует описанные в этом разделе концепции.

Пример использования событий объекта `Chart` во встроенной диаграмме

Пример, рассматриваемый в этом разделе, призван закрепить материал, приведенный выше. На рис. 18.14 отображена встроенная диаграмма, которая работает подобно карте изображения. На ее областях можно щелкать для получения разных результатов. Щелчок на одном из столбцов диаграммы приводит к активизации того рабочего листа, на котором содержатся подробные данные об этой области диаграммы.

Рис. 18.14. Эта диаграмма может использоваться в качестве карты изображения

Рабочая книга состоит из четырех листов. Лист `Главный лист` содержит встроенную диаграмму. Другие листы называются `Север`, `Юг` и `Запад`. Формулы в диапазоне B1:B4 суммируют данные в соответствующих листах, а итоговые данные отображаются на диаграмме. Щелчок на столбце диаграммы приводит к возникновению события. Процедура обработки этого события активизирует соответствующий лист, чтобы пользователь мог просмотреть подробные сведения выбранного диапазона.

В рабочей книге расположены модуль класса, который называется `EmbChartClass`, а также обычный модуль VBA — `Module1`. В демонстрационных целях рабочий лист `Главный лист` содержит флажок (из группы **Формы** (Forms)). Установка этого флажка приводит к вызову процедуры `CheckBox1_Click`, которая активизирует/отключает мониторинг событий.

Кроме того, на каждом из других рабочих листов находится кнопка, которая вызывает макрос ReturntoMain, повторно активизирующий лист Главный лист.

Ниже приводится полный листинг модуля Module1.

```
Dim SummaryChart As New EmbChartClass
Sub CheckBox1_Click()
   If Worksheets("Главный лист").CheckBoxes("Флажок 1") = xlOn Then
      ' Вызов событий диаграммы
      Range("A1").Select
      Set SummaryChart.myChartClass =
         Worksheets(1).ChartObjects(1).Chart
   Else
      ' Отключение событий диаграммы
      Set SummaryChart.myChartClass = Nothing
      Range("A1").Select
   End If
End Sub
Sub ReturnToMain()
'  Вызывается с помощью кнопки на рабочем листе
   Sheets("Главный лист").Activate
End Sub
```

Первая инструкция объявляет новую объектную переменную SummaryChart, которая имеет тип EmbChartClass (как вы помните, это название модуля класса). Как только пользователь устанавливает флажок **События диаграммы** (Enable Chart Events), внедренная диаграмма присваивается объекту SummaryChart, который активизирует события для диаграммы. Ниже приводится содержимое модуля класса для модуля класса EmbChartClass.

```
Public WithEvents myChartClass As Chart
Private Sub myChartClass_MouseDown(ByVal Button As Long, _
   ByVal Shift As Long, ByVal X As Long, ByVal Y As Long)

   Dim IDnum As Long
   Dim a As Long, b As Long

'  Следующий оператор возвращает значения для
'  IDnum, a и b
   myChartClass.GetChartElement X, Y, IDnum, a, b
'  На каком ряду был выполнен щелчок мышью?
   If IDnum = xlSeries Then
      Select Case b
         Case 1
            Sheets("Север").Activate
         Case 2
            Sheets("Юг").Activate
         Case 3
            Sheets("Запад").Activate
      End Select
   End If
   Range("A1").Select
End Sub
```

После щелчка на диаграмме вызывается событие MouseDown, которое, в свою очередь, вызывает процедуру myChartClass_MouseDown. Эта процедура использует метод GetChartElement для определения элемента диаграммы, на котором произведен щелчок мышью. Метод GetChartElement возвращает информацию об элементе диаграммы

в виде его координат X и Y (доступ к этой информации обеспечивается через аргументы процедуры myChartClass_MouseDown).

Компакт-диск

Рассматриваемая в этом разделе рабочая книга находится на прилагаемом компакт-диске в файле chart image map.xlsm.

Тонкости создания диаграмм

В этом разделе вы найдете приемы работы с диаграммами, которые я вырабатывал на протяжении ряда лет. Одни из них можно применить при разработке приложений, другие пригодны в развлекательных целях. И наконец, изучив их, вы получите дополнительную информацию об объектной модели диаграмм.

Печать встроенных диаграмм на всю страницу

При выделении встроенной диаграммы можно распечатать диаграмму с помощью команды Файл⇨Печать (File⇨Print). Встроенная диаграмма будет распечатана на всю страницу (как будто она находится на листе диаграммы), но при этом останется встроенной. Следующий макрос печатает все встроенные диаграммы на активном листе, причем каждая диаграмма печатается на всю страницу.

```
Sub PrintEmbeddedCharts()
   Dim ChtObj As ChartObject
   For Each ChtObj In ActiveSheet.ChartObjects
      ChtObj.Chart.Print
   Next ChtObj
End Sub
```

Отображение/сокрытие рядов данных

По умолчанию на диаграммах Excel не отображаются данные, находящиеся в скрытых строках или столбцах. Рабочая книга, показанная на рис. 18.15, демонстрирует простой способ сокрытия/отображения пользователем избранных рядов данных. Диаграмма включает семь рядов данных, которые "перемешаны в живописном беспорядке". Благодаря относительно простому макросу пользователь может обратиться к элементу управления ActiveX CheckBox для выбора просматриваемого ряда данных. На рис. 18.16 показана диаграмма с тремя отображаемыми рядами данных.

Каждый ряд данных находится в именованном диапазоне: Product_A, Product_B и т.д. Для каждого флажка имеется собственная процедура обработки события Click. Например, приведенная ниже процедура вызывается после того, как пользователь установит флажок Продукт А.

```
Private Sub CheckBox1_Click()
   ActiveSheet.Range("Product_A").EntireColumn.Hidden = _
      Not ActiveSheet.OLEObjects(1).Object.Value
End Sub
```

Компакт-диск

Рассматриваемая в этом разделе рабочая книга находится на прилагаемом компакт-диске в файле hide and unhide series.xlsm.

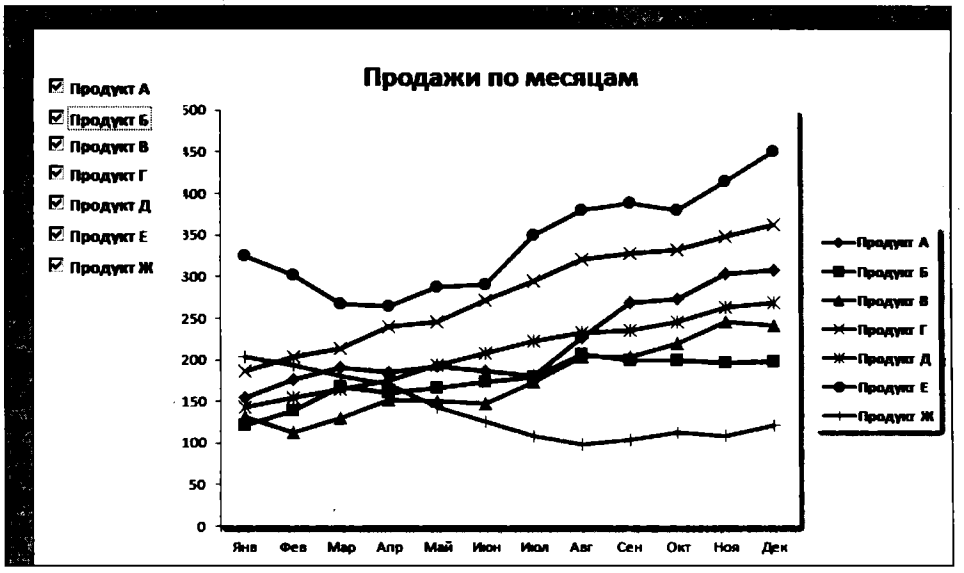


Рис. 18.15. Выберите отображаемые ряды данных с помощью соответствующих флажков

Рис. 18.16. После сокрытия некоторых столбцов данных диаграмма стала более понятной

Создание фиксированной диаграммы

Как правило, диаграмма Excel использует данные, которые хранятся в диапазоне. Изменение данных в диапазоне приводит к автоматическому обновлению содержимого диаграммы. В некоторых случаях возникает необходимость "отсоединить" диаграмму от ее диапазона данных и создать *фиксированную диаграмму* (чтобы она никогда не изменялась). Например, если данные диаграммы создаются рядом сценариев "что если", то

может возникнуть необходимость в сохранении базовой диаграммы, которая сравнивается с результатами выполнения других сценариев.

Существует три метода создания такой диаграммы.

- **Вставка диаграммы в виде изображения.** Активизируйте диаграмму и выберите команду Главная⇨Буфер обмена⇨Копировать⇨Копировать как рисунок (Home⇨Clipboard⇨Copy⇨Copy As Picture). Примите все установки, заданные по умолчанию, в окне Копировать рисунок (Copy Picture). Щелкните мышью на ячейке и выберите команду Главная⇨Буфер обмена⇨Вставить (Home⇨Clipboard⇨Paste). В результате будет вставлена картинка, которая является скопированной диаграммой.
- **Преобразование ссылки на диапазон в массивы значений.** Щелкните мышью на ряде данных, после чего щелкните на панели формул. Нажмите клавишу <F9> для преобразования диапазонов в массив. Повторите эти действия для каждого ряда данных диаграммы.
- **Использование макроса для присвоения массива вместо диапазона свойствам `Xvalues` либо `Values` объекта `Series`.** Эта методика описана ниже.

Представленная ниже процедура создает диаграмму (рис. 18.17) на основе данных, взятых из нескольких массивов, причем информация на рабочем листе не сохраняется. Более того, функция РЯД (SERIES) оперирует массивами, а не ссылками на диапазоны.

Рис. 18.17. На этой диаграмме использованы данные из массивов (они не хранятся в рабочем листе)

```
Sub CreateUnlinkedChart()
   Dim MyChart As Chart
   Set MyChart = ActiveSheet.Shapes.AddChart.Chart
   With MyChart
      .SeriesCollection.NewSeries
```

```
      .SeriesCollection(1).Name = "Продажи"
      .SeriesCollection(1).XValues = Array("Январь", _
         "Февраль", "Март")
      .SeriesCollection(1).Values = Array(125, 165, 189)
      .ChartType = xlColumnClustered
      .SetElement msoElementLegendNone
   End With
End Sub
```

Поскольку в Excel существует ограничение на длину функции РЯД, эта методика может применяться только для обработки относительно небольших наборов данных.

Приведенная ниже процедура создает изображение, которое представляет собой копию активной диаграммы (при этом исходная диаграмма не удаляется). Код работает только с внедренными диаграммами.

```
Sub ConvertChartToPicture()
   Dim Cht As Chart
   If ActiveChart Is Nothing Then Exit Sub
   If TypeName(ActiveSheet) = "Chart" Then Exit Sub
   Set Cht = ActiveChart
   Cht.CopyPicture Appearance:=xlPrinter, _
      Size:=xlScreen, Format:=xlPicture
   ActiveWindow.RangeSelection.Select
   ActiveSheet.Paste
End Sub
```

После преобразования диаграммы в рисунок можно создавать интересные эффекты, обратившись к команде Работа с рисунками⇨Формат⇨Стили рисунков (Picture Tools⇨Format⇨Picture Styles). Примеры подобных эффектов представлены на рис. 18.18.

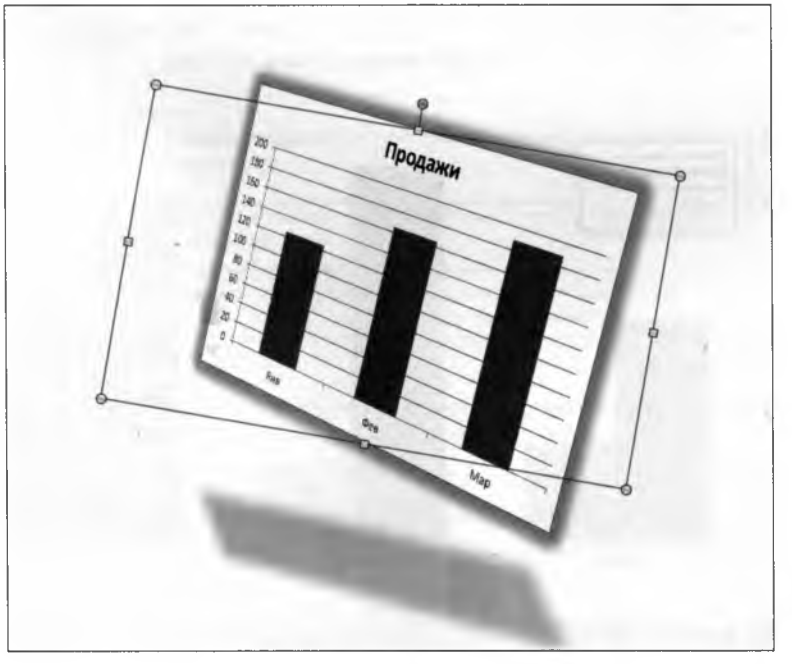


Рис. 18.18. После преобразования диаграммы в рисунок можно выполнять его дальнейшую обработку с помощью целого ряда команд

Компакт-диск

Два примера, рассматриваемые в этом разделе, можно найти на прилагаемом к книге компакт-диске в файле `unlinked charts.xlsm`.

Отображение подсказки

В диаграммах очень удобно использовать экранные подсказки. *Экранные подсказки* — это небольшие сообщения, которые отображаются на экране при наведении указателя мыши на элемент диаграммы. Обычно в экранной подсказке представлены имя элемента (для рядов данных) и значение точки данных. Объект `Chart` не позволяет управлять экранными подсказками для диаграммы.

Совет

Для активизации/отключения экранной подсказки выполните команду Файл⇨Параметры Excel (File⇨Excel Options) для отображения диалогового окна Параметры Excel (Excel Options). Выберите вкладку Дополнительно (Advanced) и найдите раздел Экран (Display). Установите флажки Показывать имена элементов диаграммы при наведении указателя (Show Chart Element Names on Hover) и Показывать значения точек данных при наведении указателя (Show Data Point Values on Hover).

В этом разделе описан способ управления экранными подсказками. На рис. 18.19 показана экранная подсказка, отображенная с помощью события `MouseOver`. При наведении указателя мыши на столбец в левом верхнем углу диаграммы появится текстовое поле с информацией о выбранной точке данных.

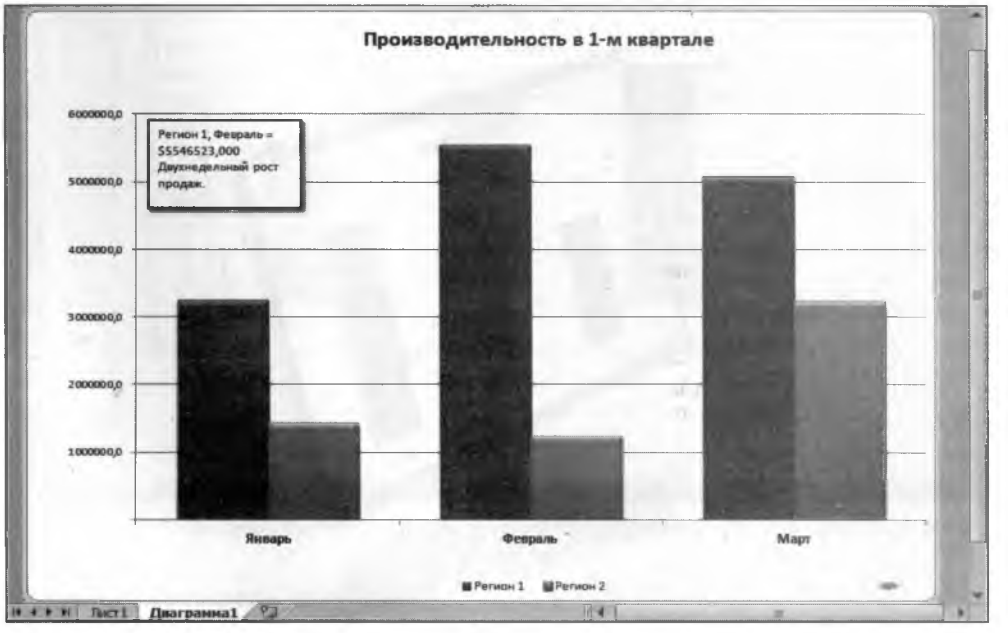


Рис. 18.19. Экранная подсказка отображает сведения о точке данных, над которой находится указатель мыши

Ниже приведена процедура обработки событий, которая находится в модуле кода для листа диаграммы.

```
Private Sub Chart_MouseMove(ByVal Button As Long, _
   ByVal Shift As Long, ByVal X As Long, ByVal Y As Long)
   Dim ElementId As Long
   Dim arg1 As Long, arg2 As Long
   Dim NewText As String
   On Error Resume Next
   ActiveChart.GetChartElement X, Y, ElementId, arg1, arg2
   If ElementId = xlSeries Then
      NewText = Sheets("Лист1").Range("Комментарий").Offset _
         (arg2, arg1)
   Else
      NewText = ""
   End If
      ActiveChart.Shapes(1).TextFrame.Characters.Text = NewText
   End Sub
```

Эта процедура отслеживает движения указателя мыши на листе диаграммы. Координаты указателя содержатся в переменных X и Y, подставляемых в процедуру. Аргументы `Button` и `Shift` в процедуре не используются.

Как и в предыдущем примере, ключевой элемент процедуры — это метод `GetChartElement`. Если переменная `ElementId` равна `xlSeries`, то указатель наведен на ряд данных. В результате отображается окно подсказки, содержащее сведения о точке данных (рис. 18.20). Если указатель мыши находится за пределами ряда данных, то текст скрывается.

	A	B	C
1	Месяц	Регион 1	Регион 2
2	Январь	3 245 151	1 434 343
3	Февраль	5 546 523	1 238 709
4	Март	5 083 204	3 224 855
5			
6	Комментарии		
7		Регион 1, Январь = $3245151,000	Регион 2, Январь = $1434343,000
8		Регион 1, Февраль = $5546523,000 Двухнедельный рост продаж.	Регион 2, Февраль = $1238709,000
9		Регион 1, Март = $5083204,000	Регион 2, Март = $3224855,000 Объединение с Лос-Анджелесом на 3 неделе.
10			

Рис. 18.20. В диапазоне B7:C9 содержится информация о точке данных, которая отображается в окне сообщения диаграммы

Пример рабочей книги также содержит процедуру `ChartActivate`, которая отключает отображение стандартной подсказки, а также процедуру `Chart_Deactivate`, которая активизирует стандартную подсказку. Ниже приведен код процедуры `Chart_Activate`.

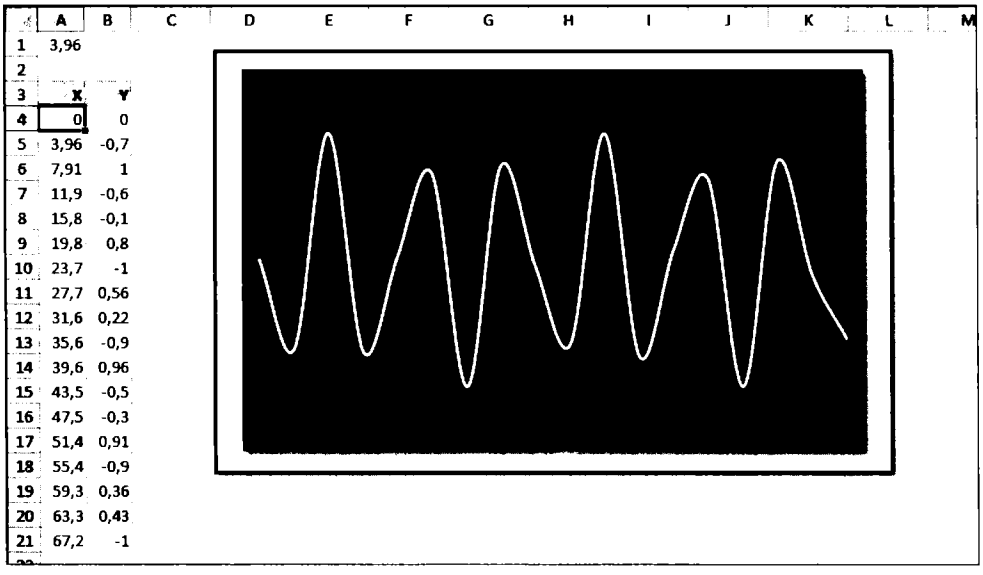
```
Private Sub Chart_Activate()
   Application.ShowChartTipNames = False
   Application.ShowChartTipValues = False
End Sub
```

Компакт-диск

На прилагаемом к книге компакт-диске находятся примеры для встроенной диаграммы (`mouseover event-embedded.xlsm`) и для листа диаграммы (`mouseover event-chart sheet.xlsm`).

Анимирование диаграмм

Программа Excel отображает простую анимацию (хотя многие пользователи не знают об этой возможности). Например, можно анимировать фигуры и графики. Рассмотрим график, показанный на рис. 18.21.



	A	B
1	3,96	
2		
3	X	Y
4	0	0
5	3,96	-0,7
6	7,91	1
7	11,9	-0,6
8	15,8	-0,1
9	19,8	0,8
10	23,7	-1
11	27,7	0,56
12	31,6	0,22
13	35,6	-0,9
14	39,6	0,96
15	43,5	-0,5
16	47,5	-0,3
17	51,4	0,91
18	55,4	-0,9
19	59,3	0,36
20	63,3	0,43
21	67,2	-1

Рис. 18.21. С помощью простой процедуры VBA на основе этого графика можно создать интересную анимационную картинку

Значения аргумента X (столбец A) зависят от значения, которое хранится в ячейке A1. Значение в каждой строке равно сумме значения в предыдущей строке и значения в ячейке A1. Столбец B содержит формулы, которые подсчитывают значение функции `SIN` для соответствующего значения в столбце A. Представленная далее простая процедура создает интересную анимационную последовательность. Она изменяет значение в ячейке A1, что приводит к изменению диапазона значений X и Y.

```
Sub AnimateChart()
   Dim i As Long
   Range("A1") = 0
   For i = 1 To 150
      DoEvents
      Range("A1") = Range("A1") + 0.035
      DoEvents
   Next i
   Range("A1") = 0
End Sub
```

Ключевым в процессе анимирования диаграмм является использование одной или более инструкций DoEvents. Эта инструкция передает контроль операционной системе, которая вызывает обновление диаграммы всякий раз, когда контроль передается обратно Excel. Если бы эти инструкции не применялись, изменения в диаграмме не отображались бы в цикле.

Компакт-диск

На прилагаемом компакт-диске находится рабочая книга, в которой содержится рассматриваемая в этом разделе анимированная диаграмма, а также несколько других анимированных диаграмм. Эта книга находится в файле animated charts.xlsm.

Прокрутка диаграммы

На рис. 18.22 показана диаграмма, которая включает 5218 точек данных для каждого ряда. Рабочая книга включает следующие шесть названий:

- НачальныйДень — имя ячейки F1;
- КоличествоДней — имя ячейки F2;
- Приращение — имя ячейки F3 (применяется для автоматической прокрутки);
- Дата — именованная формула:
  =СМЕЩ(Лист1!$A$1,НачальныйДень,0, КоличествоДней,1)
- Продукт_А — именованная формула:
  =СМЕЩ(Лист1!$B$1,НачальныйДень,0, КоличествоДней,1)
- Продукт_Б — именованная формула:
  =СМЕЩ(Лист1!$C$1,НачальныйДень,0, КоличествоДней,1)

Каждая из функций РЯД (SERIES) в диаграмме использует имена для значений категорий и данных. Ниже приведена функция РЯД для ряда данных Продукт_А (для лучшего понимания было удалено имя листа и книги).

```
=РЯД($B$1,Дата,Продукт_А,1)
```

А вот функция РЯД для ряда данных Продукт_Б:

```
=РЯД($C$1,Дата,Продукт_Б,2)
```

Благодаря использованию этих имен можно указать значения параметров НачальныйДень и КоличествоДней, после чего диаграмма отобразит поднабор данных.

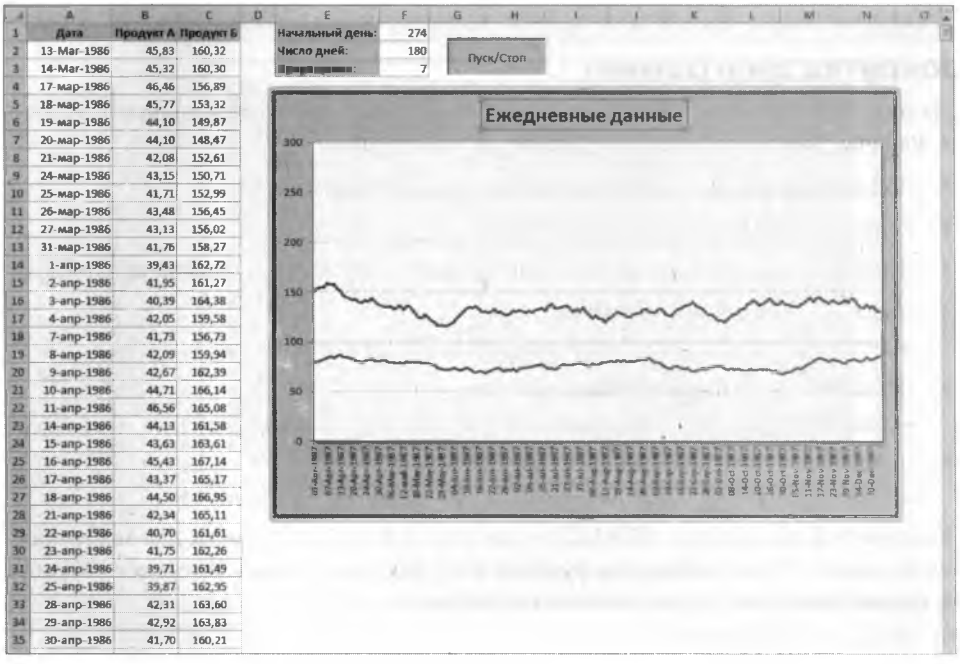
Компакт-диск

На прилагаемом компакт-диске находится рабочая книга, в которой содержится рассматриваемый в этом разделе пример, а также ряд других примеров анимированных диаграмм. Файл примера называется scrolling chart.xlsm.

Прокрутка диаграммы осуществляется с помощью следующего довольно простого макроса. Для запуска этого макроса используется кнопка, находящаяся на рабочем листе.

```
Public AnimationInProgress As Boolean

Sub AnimateChart()
   Dim StartVal As Long, r As Long
   If AnimationInProgress Then
```

```
        AnimationInProgress = False
        End
    End If
    AnimationInProgress = True
    StartVal = Range("НачальныйДень")
    For r = StartVal To 5219 - Range("КоличествоДней") _
        Step Range("Приращение")
        Range("НачальныйДень") = r
        DoEvents
    Next r
    AnimationInProgress = False
End Sub
```



	Дата	Продукт А	Продукт Б
2	13-Мар-1986	45,83	160,32
3	14-Мар-1986	45,32	160,30
4	17-мар-1986	46,46	156,89
5	18-мар-1986	45,77	153,32
6	19-мар-1986	44,10	149,87
7	20-мар-1986	44,10	148,47
8	21-мар-1986	42,08	152,61
9	24-мар-1986	43,15	150,71
10	25-мар-1986	41,71	152,99
11	26-мар-1986	43,48	156,45
12	27-мар-1986	43,13	156,02
13	31-мар-1986	41,76	158,27
14	1-апр-1986	39,43	162,72
15	2-апр-1986	41,95	161,27
16	3-апр-1986	40,39	164,38
17	4-апр-1986	42,05	159,58
18	7-апр-1986	41,73	156,73
19	8-апр-1986	42,09	159,94
20	9-апр-1986	42,67	162,39
21	10-апр-1986	44,71	166,14
22	11-апр-1986	46,56	165,08
23	14-апр-1986	44,13	161,58
24	15-апр-1986	43,63	163,61
25	16-апр-1986	45,43	167,14
26	17-апр-1986	43,37	165,17
27	18-апр-1986	44,50	166,95
28	21-апр-1986	42,34	165,11
29	22-апр-1986	40,70	161,61
30	23-апр-1986	41,75	162,26
31	24-апр-1986	39,71	161,49
32	25-апр-1986	39,87	162,95
33	28-апр-1986	42,31	163,60
34	29-апр-1986	42,92	163,83
35	30-апр-1986	41,70	160,21

Рис. 18.22. Значения в столбце F определяют данные, отображаемые в диаграмме

Процедура `AnimateChart` использует глобальную переменную (`AnimationInProgress`) для отслеживания состояния анимации. Анимация выполняется в цикле, который изменяет значение в ячейке `НачальныйДень`. Поскольку это значение используется двумя рядами данных, диаграмма постоянно обновляется с применением нового начального значения. Установка `Приращение` определяет скорость прокрутки диаграммы.

Для остановки анимации использовалась инструкция `End` (вместо традиционной в таких случаях инструкции `Exit Sub`). Подобное решение объясняется тем, что инструкция `Exit Sub` иногда вызывает ненадежную работу Excel.

Создание диаграммы с графиком гипоциклоиды

Даже если в школе тригонометрия вызывала у вас трудности, пример, представленный в этом разделе, вам точно понравится (он основан на тригонометрических функциях). Рабочая книга, показанная на рис. 18.23, содержит бесконечное количество гипоцик-

лоидных кривых. Гипоциклоида — это траектория точки, находящейся на окружности, которая движется внутри другой окружности. Подобный прием использовался в популярной детской игре "Спирограф".

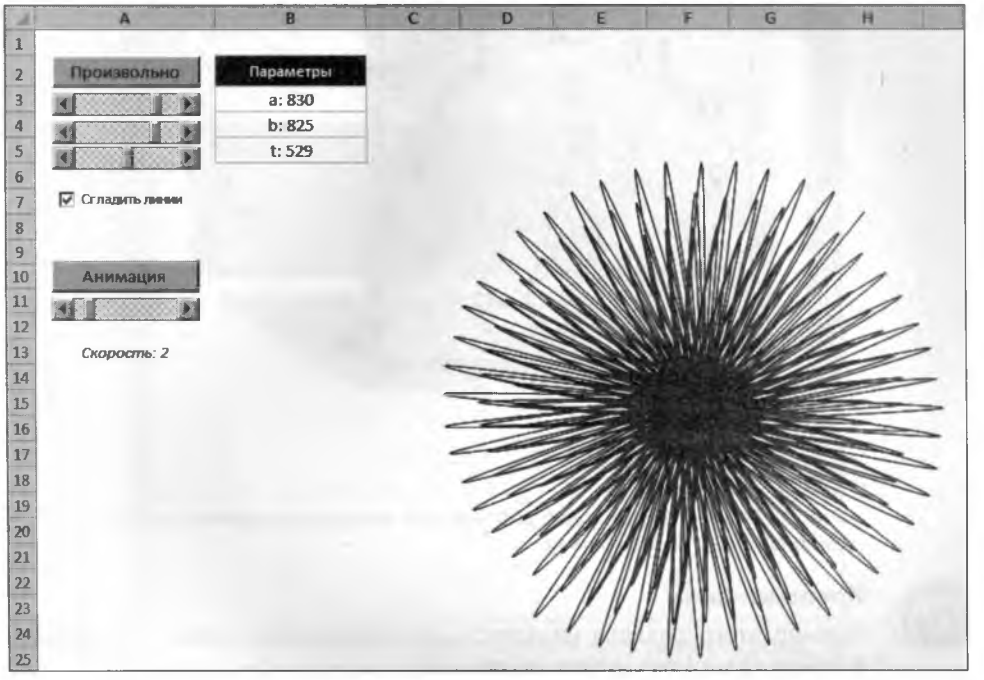


Рис. 18.23. Эта рабочая книга генерирует бесконечное количество гипоциклоидных кривых

Компакт-диск

Рассматриваемый в этом разделе пример находится на прилагаемом к книге компакт-диске в файле `hypocycloid-animate.xlsm`.

В данном случае диаграмма представляет собой график, все элементы которого скрыты (за исключением рядов данных). Данные по осям абсцисс и ординат генерируются с помощью формул, находящихся в столбцах A и B. Элементы управления полосы прокрутки, находящиеся вверху слева, обеспечивают настройку трех параметров, с помощью которых определяется внешний вид диаграммы. Кроме того, щелчком на кнопке Произвольно вы задаете генерирование случайных значений для этих трех параметров. Рассмотренная в этом разделе диаграмма интересна сама по себе, но еще интереснее она выглядит, будучи анимированной. Анимация достигается путем изменения начального значения ряда данных, которое осуществляется в цикле.

Создание диаграммы часов

На рис. 18.24 показан график, который видоизменен так, чтобы имитировать циферблат часов. Такая диаграмма не только выглядит, как часы, но и работает, как часы. Сложно придумать область применения такой диаграммы, но ее создание — хорошее практическое упражнение.

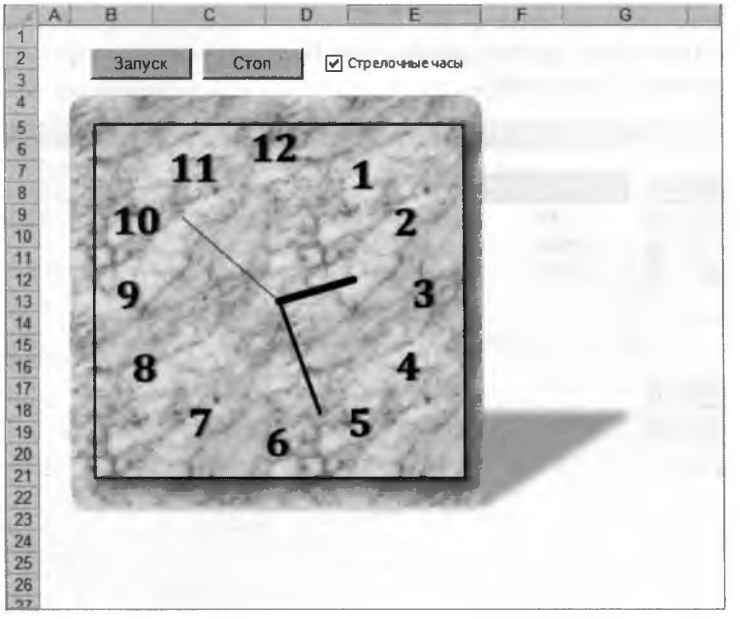


Рис. 18.24. Эти превосходно работающие часы представляют собой видоизмененную диаграмму

Компакт-диск

Пример этого раздела находится на прилагаемом к книге компакт-диске в файле `vba clock chart.xlsm`.

Кроме диаграммы часов, рабочая книга содержит текстовое поле, которое отображает время в виде строки, как показано на рис. 18.25. Обычно это поле скрыто, но его можно отобразить, сбросив флажок **Стрелочные часы**.

При рассмотрении рабочей книги, которая содержится на компакт-диске, обратите внимание на несколько следующих моментов.

- Объект `ChartObject` называется `ClockChart` и использует диапазон `DigitalClock` для цифрового отображения времени.
- Две кнопки на листе добавлены с панели инструментов **Формы** (Forms). Каждой кнопке назначена процедура (`StartClock` и `StopClock`).
- Элемент управления `CheckBox` (Флажок) (с именем `cbClockType`) на листе взят с панели инструментов **Формы** (Forms), а не с панели инструментов **Элементы управления** (Control Toolbox). Щелчок на этом объекте приводит к выполнению процедуры `cbClockType_Click`, которая изменяет значение свойства `Visible` объекта `ClockChart`. Когда данный объект становится невидимым, пользователю открывается цифровой индикатор времени.
- В качестве диаграммы используется график с четырьмя объектами `Series`. Эти ряды данных представляют часовую стрелку, минутную стрелку, секундную стрелку, а также 12 цифр.
- Процедура `UpdateClock` выполняется после щелчка на кнопке **Запуск**. Она использует метод `OnTime` объекта `Application`. Этот метод позволяет выполнять

процедуру в определенное время. Перед завершением процедуры `UpdateClock` устанавливается новое событие `OnTime`, которое произойдет через одну секунду. Другими словами, процедура `UpdateClock` вызывается каждую секунду.

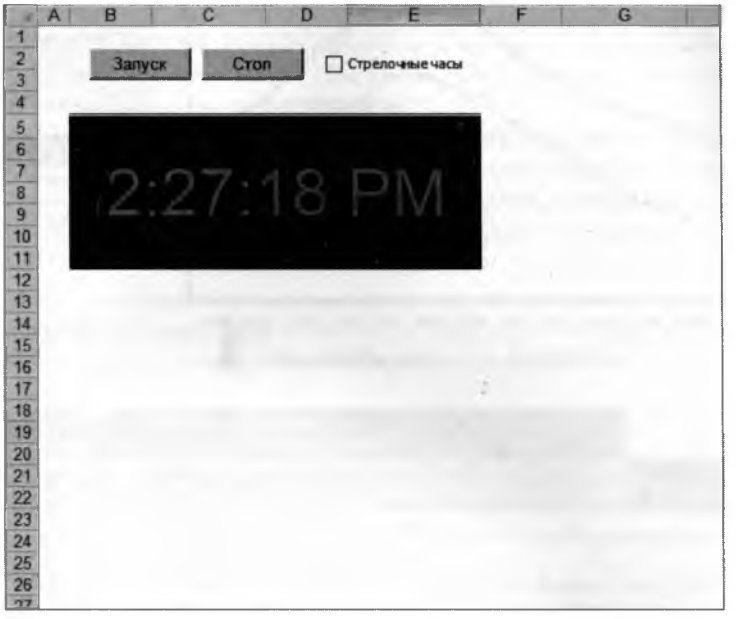


Рис. 18.25. Отобразить на листе цифровые часы намного проще, но результат не будет столь эффектным, как в случае стрелочных часов

- В отличие от многих других диаграмм, эта диаграмма не использует диапазон ячеек рабочего листа для хранения собственных данных. Вместо этого необходимые данные рассчитываются с помощью кода VBA и передаются непосредственно в свойства `Values` и `XValues` объектов `Series` диаграммы.

Предупреждение

Диаграмма часов — интересный пример использования макросов VBA, но вряд ли ее можно применять для встраивания часов в рабочий лист. Если поступить подобным образом, макрос VBA будет постоянно выполняться в фоновом режиме, взаимодействуя с другими макросами, что приведет к снижению производительности системы.

Создание интерактивной диаграммы без написания макросов

В этом примере (рис. 18.26) вашему вниманию представлено приложение, которое позволяет отображать для выбранного города США (всего — 284 города) диаграмму среднемесячной температуры, нормы осадков, средней скорости ветра и среднего количества солнечных дней в году.

	Средняя температура											
	Янв	Фев	Мар	Апр	Май	Июн	Июл	Авг	Сен	Окт	Ноя	Дек
TUCSON, AZ	51,7	55,0	59,2	66,0	74,5	84,1	86,5	84,9	80,9	70,5	58,7	51,9
MISSOULA, MT	23,5	29,0	37,6	45,2	52,7	60,2	66,9	66,3	56,1	44,4	32,0	23,4
Разница	28,2	26,0	21,6	20,8	21,8	23,9	19,6	18,6	24,8	26,1	26,7	28,5

Рис. 18.26. В этом приложении используется несколько способов построения графиков, отображающих климатические данные для двух выбранных американских городов

Самое интересное заключается в том, что для решения этой задачи не используются макросы VBA. Интерактивность приложения обеспечивается встроенными средствами Excel. Город выбирается в раскрывающемся списке, он проверяется на принадлежность к списку (с помощью функции проверки данных), после чего указывается тип графика с помощью одного из четырех переключателей. Работа приложения сводится к выполнению всего нескольких формул.

В этом примере показано, что удобное и практичное приложение можно создать без использования макросов.

Компакт-диск

Рассмотренная в этом разделе рабочая книга находится на прилагаемом компакт-диске в файле `climate data.xlsx`.

Далее описаны действия, выполняемые для разработки этого приложения.

Получение данных приложения

Всего пять минут у меня ушло на поиск в Интернете исходных данных, требуемых для создаваемого приложения. Эти данные были найдены на сайте Национального центра метеорологических данных (National Climatic Data Center). Затем я скопировал данные в окно браузера, вставил их на рабочий лист Excel и выполнил окончательную "доводку". В результате были получены четыре таблицы из 13 столбцов, которые были названы `PrecipitationData`, `TemperatureData`, `SunshineData` и `WindData`. Чтобы не загромождать экран данными, я поместил их на отдельном листе (под названием `Data`).

Создание переключателей на рабочем листе

Необходимо предоставить пользователям возможность выбрать тип графика, который будет отображаться на экране. Один из наилучших вариантов — это добавление элементов управления **Переключатель** (OptionButton) с панели инструментов **Формы** (Forms). Поскольку переключатели объединяются в группы, все они ссылаются на ячейку O3. Эта ячейка тем не менее может принимать несколько значений в диапазоне от 1 до 4 в зависимости от активного переключателя.

Далее необходимо реализовать механизм выбора таблицы данных на основе активного переключателя или значения в ячейке O3. В результате поиска решения мне удалось написать формулу, вводимую в ячейку O4, в которой используется функция ВЫБОР.

```
=ВЫБОР(O3,"TemperatureData","PrecipitationData","SunshineData",_
      "WindData")
```

Таким образом, в ячейке O4 отображается название одного из представляемых на диаграмме диапазонов данных. Для визуального выделения переключателей на рабочем листе я немного изменил форматирование прилегающих ячеек.

Создание списка городов

Следующий этап — создание раскрывающегося списка, в котором пользователь сможет выбрать город с отображаемой для него информацией о погоде. Упрощает эту процедуру специальное средство проверки данных Excel. Но сначала придется выполнить предварительную обработку данных. Объединим ячейки J11:M11 для первого города и назовем их `City1`. Затем объединим ячейки J13:M13 и назовем их `City2`.

Для упрощения дальнейших действий я создал именованный диапазон `CityList`, который соответствует первому столбцу таблицы `PrecipitationData`.

Выполните приведенные ниже действия для создания раскрывающегося списка.

1. Выберите диапазон ячеек J11:M11. (Помните о том, что эти ячейки — объединенные.)
2. Выберите команду **Данные⇨Проверка данных** (Data⇨Data Validation) для отображения диалогового окна Excel **Проверка вводимых значений** (Data Validation).
3. В окне **Проверка вводимых значений** выберите вкладку **Параметры** (Settings).
4. В поле со списком **Тип данных** (Allow) выберите значение **Список** (List).
5. В поле **Источник** (Source) введите **`=CityList`**.
6. Щелкните на кнопке **OK**.
7. Скопируйте диапазон J11:M11 в диапазон J13:M13. В результате будут продублированы установки проверки вводимых значений для второго города.

Результат показан на рис. 18.27.

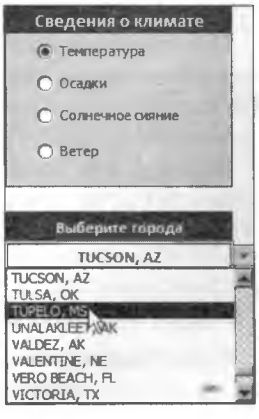


Рис. 18.27. Воспользуйтесь раскрывающимся списком проверки вводимых значений для выбора города

Создание диапазона данных для интерактивной диаграммы

Ключевой момент в этой диаграмме заключается в следующем: диаграмма строится на основе определенного диапазона данных. Данные в этом диапазоне получены из соответствующей таблицы данных с помощью формул, в которых фигурирует функция ВПР (VLOOKUP), как показано на рис. 18.28.

	A	B	C	D	E	F	G	H	I	J	K	L	M
20													
21		Средняя температура											
22		Янв	Фев	Мар	Апр	Май	Июн	Июл	Авг	Сен	Окт	Ноя	Дек
23	TUCSON, AZ	51,7	55,0	59,2	66,0	74,5	84,1	86,5	84,9	80,9	70,5	58,7	51,9
24	MISSOULA, MT	23,5	29,0	37,6	45,2	52,7	60,2	66,9	66,3	56,1	44,4	32,0	23,4
25	Разница	28,2	26,0	21,6	20,8	21,8	23,9	19,6	18,6	24,8	26,1	26,7	28,5
26													

Рис. 18.28. При построении этой диаграммы используются данные, выбираемые с помощью формул в диапазоне A23:M24

В ячейке A23 расположена формула, которая просматривает данные на основе содержимого ячейки `City1`.

```
=ВПР(City1,ДВССЫЛ(DataTable),СТОЛБЕЦ(),ЛОЖЬ)
```

Формула в ячейке A24 совпадает с формулой в ячейке A24, за исключением того, что она просматривает данные на основе содержимого ячейки `City2`.

```
=ВПР(City2,ДВССЫЛ(DataTable),СТОЛБЕЦ(),ЛОЖЬ)
```

После введения этих формул я просто скопировал их в остальные 12 столбцов.

Примечание

Вы можете быть удивлены тем фактом, что в качестве третьего аргумента функции ВПР используется функция СТОЛБЕЦ (COLUMN). Она возвращает номер столбца с ячейкой, содержащей формулу. Это позволяет избежать жесткого кодирования выбираемого столбца, что, в свою очередь, позволит использовать одну и ту же формулу в любом столбце.

Под строками с данными, относящимися к двум городам, находится строка формул, ежемесячно вычисляющих разницу между климатическими показателями этих городов. Для выделения большей (меньшей) разницы используются различные цвета, получаемые с помощью условного форматирования.

Подпись над названиями месяцев генерируется формулой, которая ссылается на ячейку `DataTable` и формирует описательный заголовок.

```
="Average " & ЛЕВСИМВ(DataTable,ДЛСТР(DataTable)-4)
```

Создание интерактивной диаграммы

После выполнения всех описанных выше задач компоновка интерактивной диаграммы не должна вызывать больших затруднений. Окончательный результат будет представлен двумя графиками, данные которых занесены в диапазон A22:M24. Заголовок диаграммы сохранен в ячейке A21. Данные в диапазоне A22:M24 изменяются в зависимости от выбранного переключателя и города в раскрывающемся списке.

Спарклайны

В заключительном разделе главы рассматриваются спарклайны — новый вид диаграмм, появившийся в Excel 2010. *Спарклайн* — это небольшая диаграмма, отображаемая в ячейке. С помощью спарклайнов можно быстро оценить тенденции, в соответствии с которыми изменяются данные. Благодаря компактности спарклайны часто объединяются в группы.

На рис. 18.29 показаны примеры типов спарклайнов, поддерживаемых в Excel.

	A	B	C	D	E	F	G	H
1	**График**							
2	***Номер детали***	***Янв***	***Фев***	***Мар***	***Апр***	***Май***	***Июн***	***Спарклайны***
3	A-13	103,98	98,92	88,12	86,34	75,58	71,2	
4	C-09	212,74	218,7	202,18	198,56	190,12	181,74	
5	K-88	75,74	73,68	69,86	60,34	64,92	59,46	
6	W-31	91,78	95,44	98,1	99,46	98,68	105,86	
7	M-03	324,48	309,14	313,1	276,24	276,24	260,9	
8								
9	**Столбец**							
10	***Номер детали***	***Янв***	***Фев***	***Мар***	***Апр***	***Май***	***Июн***	***Спарклайны***
11	A-13	103,98	98,92	88,12	86,34	75,58	71,2	
12	C-09	212,74	218,7	202,18	198,56	190,12	181,74	
13	K-88	75,74	73,68	69,86	60,34	64,92	59,46	
14	W-31	91,78	95,44	98,1	99,46	98,68	105,86	
15	M-03	324,48	309,14	313,1	276,24	276,24	260,9	
16								
17	**Выигрыш/проигрыш**							
18	***Номер детали***	***Янв***	***Фев***	***Мар***	***Апр***	***Май***	***Июн***	***Спарклайны***
19	A-13	103,98	98,92	88,12	86,34	75,58	71,2	
20	C-09	212,74	218,7	202,18	198,56	190,12	181,74	
21	K-88	75,74	73,68	69,86	60,34	64,92	59,46	
22	W-31	91,78	95,44	98,1	99,46	98,68	105,86	
23	M-03	324,48	309,14	313,1	276,24	276,24	260,9	
24								
25								

Рис. 18.29. Примеры спарклайнов

Я рад вам сообщить о том, что Microsoft добавила объекты, соответствующие спарклайнам, в объектную модель Excel. Благодаря этому появилась возможность работать со спарклайнам с помощью средств VBA. В верхней части иерархии объектов находится коллекция `Sparkline Groups`, представляющая собой набор всех объектов `SparklineGroup`. Объект `SparklineGroup` включает объекты `Sparkline`. Вопреки ожидаемому предком коллекции `SparklineGroups` является объект `Range`, а не объект `Worksheet`. Поэтому выполнение следующего оператора приведет к появлению ошибки:

```
MsgBox ActiveSheet.SparklineGroups.Count
```

Вместо него следует использовать свойство `Cells` (возвращает объект диапазона).

```
MsgBox Cells.SparklineGroups.Count
```

В результате выполнения следующего примера кода перечисляются адреса группы спарклайнов, находящихся на активном рабочем листе.

```
Sub ListSparklineGroups()
   Dim sg As SparklineGroup
   Dim i As Long
   For i = 1 To Cells.SparklineGroups.Count
      Set sg = Cells.SparklineGroups(i)
      MsgBox sg.Location.Address
   Next i
End Sub
```

По ряду причин не следует использовать конструкцию `For Each` для обхода объектов в коллекции `SparklineGroups`. Для ссылки на объекты воспользуйтесь номерами индексов.

Ниже приводится еще один пример кода, демонстрирующий работу со спарклайнами в VBA. Процедура `SparklineReport` выводит сведения о каждом объекте спарклайна, находящемся на активном рабочем листе.

```
Sub SparklineReport()
   Dim sg As SparklineGroup
   Dim sl As Sparkline
   Dim SGType As String
   Dim SLSheet As Worksheet
   Dim i As Long, j As Long, r As Long

   If Cells.SparklineGroups.Count = 0 Then
      MsgBox "На активном листе спарклайны отсутствуют."
      Exit Sub
   End If

   Set SLSheet = ActiveSheet
'  Вставка нового рабочего листа отчета
   Worksheets.Add

'  Заголовки
   With Range("A1")
      .Value = "Отчет о спарклайне: " & SLSheet.Name & " в " _
         & SLSheet.Parent.Name
      .Font.Bold = True
      .Font.Size = 16
   End With
   With Range("A3:F3")
      .Value = Array("Группа #", "Диапазон группы _
         спарклайнов", "# в группе", "Тип", _
         "Sparkline #", "Исходный диапазон")
      .Font.Bold = True
   End With
   r = 4

'  Обход каждой группы спарклайнов
   For i = 1 To SLSheet.Cells.SparklineGroups.Count
      Set sg = SLSheet.Cells.SparklineGroups(i)
      Select Case sg.Type
         Case 1: SGType = "График"
         Case 2: SGType = "Столбец"
         Case 3: SGType = "Выигрыш/проигрыш"
      End Select

'     Обход каждого спарклайна в группе
      For j = 1 To sg.Count
         Set sl = sg.Item(j)
```

```
            Cells(r, 1) = i '# группы
            Cells(r, 2) = sg.Location.Address
            Cells(r, 3) = sg.Count
            Cells(r, 4) = SGType
            Cells(r, 5) = j '# спарклайна в группе
            Cells(r, 6) = sl.SourceData
            r = r + 1
        Next j
        r = r + 1
    Next i
ıd Sub
```

На рис. 18.30 показан отчет, сгенерированный для группы спарклайнов.

Компакт-диск

Рабочая книга `sparkline report.xlsm` находится на прилагаемом компакт-диске.

	A	B	C	D	E	F	G
1	**Отчет о спарклайнах: Лист1 в sparkline report.xlsm**						
2							
3	# группы	Диапазон группы с	# в группе	Тип	# спарклайна	Исходный диапазон	
4	1	$N$22:$N$31	10	График	1	B22:M22	
5	1	$N$22:$N$31	10	График	2	B23:M23	
6	1	$N$22:$N$31	10	График	3	B24:M24	
7	1	$N$22:$N$31	10	График	4	B25:M25	
8	1	$N$22:$N$31	10	График	5	B26:M26	
9	1	$N$22:$N$31	10	График	6	B27:M27	
10	1	$N$22:$N$31	10	График	7	B28:M28	
11	1	$N$22:$N$31	10	График	8	B29:M29	
12	1	$N$22:$N$31	10	График	9	B30:M30	
13	1	$N$22:$N$31	10	График	10	B31:M31	
14							
15	2	$N$9:$N$18	10	Столбец	1	B9:M9	
16	2	$N$9:$N$18	10	Столбец	2	B10:M10	
17	2	$N$9:$N$18	10	Столбец	3	B11:M11	
18	2	$N$9:$N$18	10	Столбец	4	B12:M12	
19	2	$N$9:$N$18	10	Столбец	5	B13:M13	
20	2	$N$9:$N$18	10	Столбец	6	B14:M14	
21	2	$N$9:$N$18	10	Столбец	7	B15:M15	
22	2	$N$9:$N$18	10	Столбец	8	B16:M16	
23	2	$N$9:$N$18	10	Столбец	9	B17:M17	
24	2	$N$9:$N$18	10	Столбец	10	B18:M18	

Рис. 18.30. Результат выполнения процедуры `SparklineReport`

Глава 19

Концепция событий Excel

В этой главе...

- Типы событий Excel
- События уровня объекта `Workbook`
- События объекта `Worksheet`
- События объекта `Chart`
- События объекта `Application`
- События объекта `UserForm`
- События, не связанные с объектами

В этой главе рассматриваются концепции событий Excel и приводится ряд примеров, которые, без сомнения, пригодятся в вашей работе.

Типы событий Excel

В нескольких предыдущих главах рассматривались примеры процедур VBA, предназначенных для обработки событий. *Процедура обработки события* — это специальная именованная процедура, которая запускается при возникновении определенного события. Простым примером может считаться процедура `CommandButton1_Click`, которая выполняется каждый раз, когда пользователь щелкает на элементе управления `CommandButton1`, находящемся в пользовательском диалоговом окне или непосредственно на рабочем листе.

Программа Excel запрограммирована на управление большим количеством событий, которые происходят в процессе работы. Эти события могут быть классифицированы следующим образом.

- **События объекта `Workbook`.** Происходят в конкретной рабочей книге. Примером такого события можно назвать события `Open` (возникает при открытии или

создании рабочей книги), BeforeSave (возникает перед сохранением рабочей книги) и NewSheet (возникает при добавлении нового листа).

- **События объекта Worksheet.** Происходят в конкретном рабочем листе. К примерам событий этого объекта можно отнести Change (изменение содержимого ячейки на листе), SelectionChange (изменение расположения курсора) и Calculate (пересчет значений рабочего листа).
- **События объекта Chart.** Имеют отношение к диаграммам. К таким событиям можно отнести Select (выделен объект на диаграмме) и SeriesChange (изменилось значение точки данных в последовательности). Для управления событиями встроенной диаграммы необходимо использовать модуль класса (см. главу 18).
- **События объекта Application.** Происходят в приложении Excel. К ним относятся NewWorkbook (создана рабочая книга), WorkbookBeforeClose (закрывается одна из рабочих книг) и SheetChange (изменено содержимое ячейки в одной из рабочих книг). Для контроля над событиями объекта Application необходимо использовать модуль класса.
- **События объекта UserForm.** Происходят в определенном диалоговом окне UserForm или в одном из объектов этого диалогового окна. Например, объект UserForm имеет событие Initialize (возникает перед отображением диалогового окна UserForm). Элемент управления CommandButton, который расположен в диалоговом окне UserForm, поддерживает событие Click (возникает после щелчка на этой кнопке).
- **События, не связанные с объектами.** Последняя категория включает события уровня приложения (объекта Application), которые называются событиями On-: OnTime и OnKey. Они работают не так, как все остальные.

Данная глава организована в соответствии с приведенным выше списком. В пределах каждого из разделов приводятся примеры, которые демонстрируют использование некоторых событий.

Понимание последовательности событий

Некоторые действия могут приводить к возникновению нескольких событий. Например, при добавлении нового листа в рабочую книгу возникают три события на уровне объекта Application:

- WorkbookNewSheet — происходит при добавлении нового листа;
- SheetDeactivate — происходит, когда отменяется выделение активного листа;
- SheetActivate — происходит, когда активизируется (выделяется) добавленный лист.

Примечание

Последовательность событий может быть намного сложнее, чем кажется на первый взгляд. Перечисленные выше события происходят на уровне объекта Application. При добавлении нового листа происходят дополнительные события на уровне объектов Workbook и Worksheet.

События возникают в определенной последовательности. Знание порядка этой последовательности может оказаться очень важным при создании процедур обработки событий. Далее будет рассматриваться порядок событий, возникающих при определенных действиях (дополнительная информация приведена в разделе "События объекта `Application`").

Размещение процедур обработки событий

Начинающие разработчики VBA-приложений часто интересуются, почему их процедуры обработки событий не запускаются при возникновении соответствующего события. Ответ чаще всего заключается в том, что процедуры расположены в неправильном месте.

В окне редактора VBE все проекты перечислены в окне Projects (Проекты). Компоненты проектов расположены в списке, показанном на рис. 19.1.

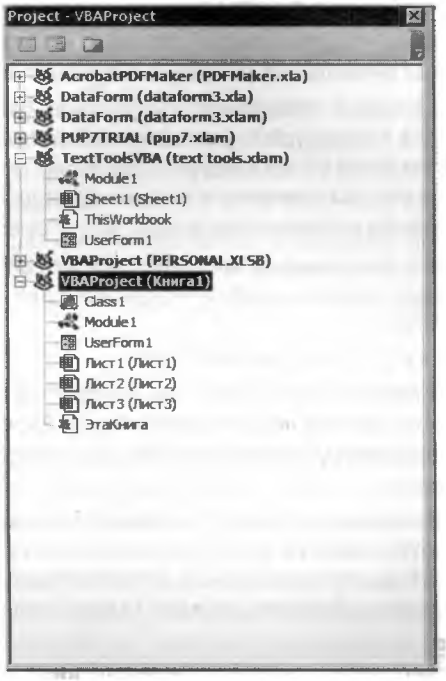


Рис. 19.1. Компоненты проекта VBA в окне Projects

Каждому из приведенных ниже компонентов соответствует отдельный модуль кода.

- **Объект `Лист` (например, `Лист1`, `Лист2` и т.д.).** Используйте этот модуль для помещения кода обработчиков событий, связанных с конкретным листом.
- **Объект `Диаграмма` (т.е. листы диаграмм).** В этом модуле находятся коды обработчиков событий, связанных с диаграммами.
- **Объект `ЭтаКнига` (`ThisWorkbook`).** В этом модуле находятся коды обработчиков событий, связанных с рабочими книгами.
- **Модули VBA общего назначения.** Процедуру обработки события никогда нельзя размещать в модуле общего назначения (необъектном модуле).

- **Модули классов.** Используйте модули классов для определения обработчиков событий специального назначения, включая события уровня приложения и события для внедренных диаграмм.

Даже если процедуры обработки событий располагаются в модулях объектов, они могут вызывать процедуры в модулях общего назначения для выполнения определенных задач. Например, следующая процедура обработки события, расположенная в модуле объекта `ЭтаКнига`, вызывает процедуру `WorkbookSetup`, которая может храниться в модуле кода VBA общего назначения.

```
Private Sub Workbook_Open()
   Call WorkbookSetup
End Sub
```

Программирование событий в предыдущих версиях Excel

Версии Excel до 97 также поддерживали события, но методы программирования процедур обработки в них отличаются от описанных в этой главе.

Например, если была создана процедура `Auto_Open`, которая хранится в модуле VBA общего назначения, эта процедура будет запускаться при каждом открытии рабочей книги. Начиная с версии Excel 97 процедура `Auto_Open` дополняется обработчиком события `Workbook_Open`, который хранится в модуле кода для объекта `ЭтаКнига`. Этот обработчик выполняется перед вызовом процедуры `Auto_Open`.

До выхода Excel 97 часто приходилось явно настраивать события. Например, для запуска процедуры каждый раз, когда в ячейку вводились данные, необходимо было выполнять следующий оператор:

```
Sheets("Лист1").OnEntry = "ValidateEntry"
```

Этот оператор инструктирует Excel о том, что следует запускать процедуру `ValidateEntry` всякий раз, когда данные вводятся в ячейку. В Excel 97 и более поздних версиях достаточно создать процедуру `Worksheet_Change`, которая будет храниться в модуле кода для объекта `Лист1`.

Из соображений совместимости Excel 97 и более поздние версии поддерживают прежний механизм обработки событий (хотя эта возможность не описана в справочной системе). Если вы разрабатываете приложения, которые предназначены для использования в Excel 97 и более старых версиях, можете смело пользоваться методами, описанными в данной главе.

Отключение событий

По умолчанию все события включены. Для их отключения выполните следующий оператор VBA:

```
Application.EnableEvents = False
```

Для включения событий воспользуйтесь следующим оператором:

```
Application.EnableEvents = True
```

Примечание

Отключить события невозможно, если они вызываются элементами управления диалогового окна `UserForm`, например события `Click`, которые генерируются после щелчка на кнопке (элемент управления `CommandButton`), находящейся в окне `UserForm`.

Итак, зачем же отключать события? Основная причина этого — для предотвращения есконечного цикла возникающих событий.

Например, ячейка A1 на рабочем листе всегда должна содержать значение, которое еньше или равно 12. Можно написать процедуру, которая будет выполняться при каж-ом изменении данных в ячейке, что позволит проверять действительность содержимого той ячейки. В данном случае контролируется событие `Change` объекта `Worksheet`, ля чего используется процедура, которая называется `Worksheet_Change`. Процедура роверяет введенные пользователем данные, и если значение больше 12, то отображает-я окно сообщения, а содержимое ячейки очищается. Проблема заключается в том, что чистка содержимого ячейки с помощью кода VBA провоцирует возникновение еще од-ого события `Change`, что приводит к повторному вызову процедуры обработки собы-ия. Поскольку подобный результат нежелателен, перед очисткой содержимого ячейки еобходимо отключить механизм возникновения событий. После выполнения операции ключите механизм поддержки событий, чтобы иметь возможность контролировать сле-ующую попытку пользователя ввести правильное значение в ячейку.

Еще одним способом предотвращения бесконечного цикла возникающих событий яв-яется объявление статической булевой переменной (`Static Boolean`) в начале про-едуры обработки события. Воспользуйтесь следующим выражением:

```
Static AbortProc As Boolean
```

Каждый раз, когда процедура должна внести изменения, необходимо присваивать пе-еменной `AbortProc` значение `True` (в противном случае потребуется убедиться, что переменной присвоено значение `False`). Вставьте следующий код в начало процедуры.

```
If AbortProc Then
   AbortProc = False
   Exit Sub
End if
```

Если происходит повторный вызов процедуры обработки события, то значение `True` переменной `AbortProc` сообщает приложению о необходимости завершения этого вы-зова. Кроме того, значение переменной `AbortProc` устанавливается равным `False`.

Перекрестная ссылка

В качестве практического примера проверки правильности введенных данных можно рассмотреть процедуру, приведенную в разделе "Проверка правильности введенных данных".

Предупреждение

Отключение событий в Excel применяется одновременно ко всем рабочим книгам. Например, если отключить события в одной процедуре, а затем открыть другую рабочую книгу, в которой существует процедура `Workbook_Open`, то эта процедура не будет выполняться.

Ввод кода процедуры обработки события

Каждая процедура обработки события имеет свое предопределенное имя. Ниже приводятся названия отдельных процедур обработки событий:

- `Worksheet_SelectionChange`;
- `Workbook_Open`;

- `Chart_Activate`;
- `Class_Initialize`.

Можно объявить процедуру обработки события вручную, но целесообразнее возложить эту задачу на редактор VBE.

На рис. 19.2 показан модуль кода объекта `ЭтаКнига` (ThisWorkbook). Для того чтобы добавить объявление процедуры, сначала выберите объект `Workbook` из списка объектов слева. Затем выберите событие из списка процедур справа. После этого будет создана "оболочка" процедуры, которая состоит из строки декларации и оператора `End Sub`.

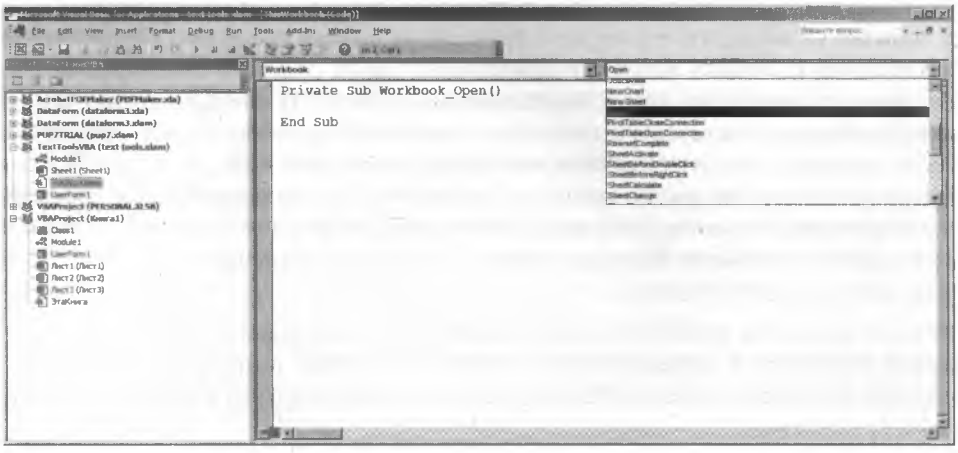


Рис. 19.2. Наилучший способ создать процедуру обработки событий — позволить VBE сделать это вместо вас

Например, если выбрать из списка объектов `Workbook`, а из списка процедур — `Open`, то редактор VBE вставит приведенную ниже (пустую) процедуру.

```
Private Sub Workbook_Open()

End Sub
```

Создаваемый код VBA должен помещаться между этими двумя строками.

Процедуры обработки событий, которые используют аргументы

Некоторые процедуры обработки событий используют набор аргументов. Например, может возникнуть необходимость в создании процедуры для обработки события `SheetActivate` рабочей книги. Если воспользоваться методикой, описанной в предыдущем разделе, то редактор VBE создаст представленную далее процедуру.

```
Private Sub Workbook_SheetActivate(ByVal Sh As Object)

End Sub
```

Эта процедура использует один аргумент (`Sh`), который представляет активизированный лист. В данном случае переменная `Sh` имеет тип `Object`, а не просто тип `Worksheet`. Это связано с тем, что активизированный лист может оказаться листом диаграммы.

Данные, которые переданы в виде аргумента, могут быть применены в коде процедуры. В приведенном ниже примере процедура выполняется при активизации рабочего листа. На экране отображаются тип и имя активизированного листа с помощью функции TypeName и свойства Name объекта, который передан в качестве аргумента.

```
Private Sub Workbook_SheetActivate(ByVal Sh As Object)
   MsgBox TypeName(Sh) & vbCrLf & Sh.Name
End Sub
```

На рис. 19.3 показано сообщение, которое отображается в случае активизации листа Лист1.

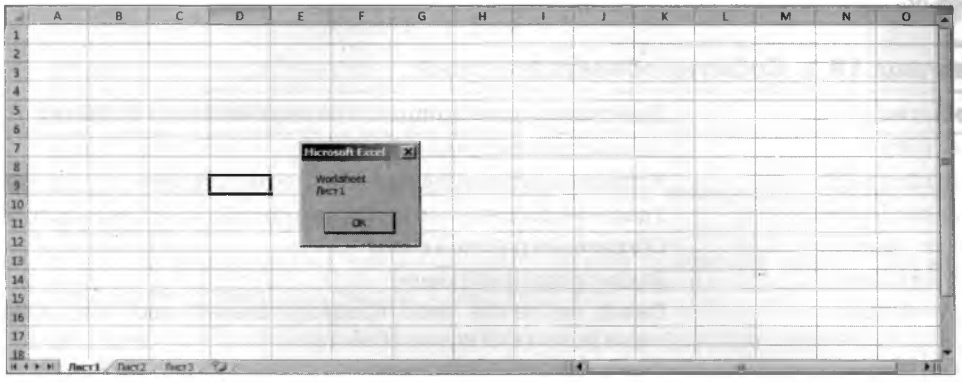


Рис. 19.3. Это окно сообщения вызывается событием SheetActivate

Некоторые процедуры обработки событий используют аргумент Cancel с типом данных Boolean. Например, объявление процедуры обработки события BeforePrint рабочей книги будет выглядеть следующим образом:

```
Private Sub Workbook_BeforePrint(Cancel As Boolean)
```

Значение аргумента Cancel, которое передается в процедуру, равно False. Но код может установить это значение равным True, что приведет к отмене печати. Следующий пример демонстрирует такую операцию.

```
Private Sub Workbook_BeforePrint(Cancel As Boolean)
   Dim Msg As String
   Dim Ans As Integer
   Msg = "Вы загрузили бумагу подходящего формата?"
   Ans = MsgBox(Msg, vbYesNo, "В процессе печати...")
   If Ans = vbNo Then Cancel = True
End Sub
```

Процедура Workbook_BeforePrint выполняется до печати рабочей книги. Эта процедура отображает окно сообщения, которое показано на рис. 19.4. Если пользователь щелкнет на кнопке Нет, то аргумент Cancel примет значение True, и ничего напечатано не будет.

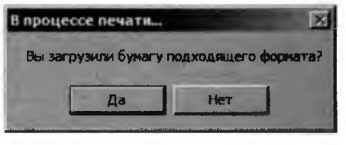


Рис. 19.4. Для отмены печати достаточно изменить значение аргумента Cancel

К сожалению, Excel не обрабатывает событие BeforePrint на уровне листа. Таким образом, код не может определить, какая часть рабочей книги будет распечатана. Зачастую можно предположить, что на печать должен выводиться лист ActiveSheet. Но не существует способа определить, что пользователь выбрал печать всей рабочей книги.

Совет

Событие `BeforePrint` также возникает перед предварительным просмотром рабочей книги.

События уровня объекта `Workbook`

События объекта `Workbook` происходят в пределах определенной рабочей книги. В табл. 19.1 перечислены события рабочей книги, а также приведено краткое описание каждого из них. Процедуры обработки событий объекта `Workbook` хранятся в модуле кода объекта `ThisWorkbook` (ЭтаКнига).

Таблица 19.1. События объекта `Workbook`

Событие	Действие, приводящее к возникновению события
`Activate`	Активизация рабочей книги
`AddinInstall`	Установка рабочей книги в качестве надстройки
`AddinUninstall`	Удаление рабочей книги, используемой в качестве надстройки
`After Save`	Сохранение рабочей книги
`BeforeClose`	Закрытие рабочей книги
`BeforePrint`	Вывод на печать либо предварительный просмотр рабочей книги (вместе со всеми объектами)
`BeforeSave`	Сохранение рабочей книги
`Deactivate`	Деактивизация рабочей книги
`NewChart`	Создание диаграммы
`NewSheet`	Создание нового листа в рабочей книге
`Open`	Открытие рабочей книги
`SheetActivate`	Активизация любого листа
`SheetBeforeDoubleClick`	Двойной щелчок на любом рабочем листе (это событие происходит до принятого по умолчанию двойного щелчка)
`SheetBeforeRightClick`	Щелчок правой кнопкой мыши на любом листе (это событие происходит до принятого по умолчанию щелчка правой кнопкой мыши)
`SheetCalculate`	Вычисление (или повторное вычисление) значений в любом рабочем листе
`SheetChange`	Изменение любого листа пользователем или внешней ссылкой
`SheetDeactivate`	Деактивизация любого листа
`SheetFollowHyperlink`	Щелчок на гиперссылке в листе
`SheetPivotTableUpdate`	Пополнение сводной таблицы новыми данными
`SheetSelectionChange`	Изменение выделения на любом листе
`WindowActivate`	Активизация любого окна рабочей книги
`WindowDeactivate`	Деактивизация любого окна рабочей книги
`WindowResize`	Изменение размера любого окна рабочей книги

Перекрестная ссылка

Если нужно проследить за событиями, возникающими для всех рабочих книг, придется работать на уровне объекта `Application` (см. раздел "События объекта `Application`"). Далее представлены примеры использования собы-

тий объекта Workbook. Все примеры процедур, представленных ниже, следует размещать в модуле кода объекта ЭтаКнига (ThisWorkbook). Если же разместить их в модуле кода другого типа, они не будут работать.

Событие Open

Одним из наиболее часто используемых является событие Open объекта WorkBook. Оно возникает при открытии рабочей книги (или надстройки). При этом выполняется процедура Workbook_Open. Эта процедура может выполнять любые действия, но чаще всего она используется для решения следующих задач:

- отображение приветственных сообщений;
- открытие других рабочих книг;
- настройка контекстных меню;
- активизация определенного листа или ячейки;
- проверка определенных условий (например, рабочая книга может требовать наличия определенной надстройки);
- установка определенных автоматических средств (например, можно определять комбинации клавиш; см. раздел "Событие OnKey");
- настройка свойства ScrollArea рабочего листа (которое не хранится в рабочей книге);
- настройка защиты UserInterfaceOnly для рабочего листа, что позволяет управлять защищенными листами в коде (этот параметр является аргументом метода Protect и не хранится в пределах рабочей книги).

Примечание

Если пользователь удерживает нажатой клавишу <Shift> при открытии рабочей книги, процедура рабочей книги Workbook_Open не вызывается. Естественно, процедура также не вызывается, если открыта рабочая книга с отключенными макросами.

Ниже приводится простой пример процедуры Workbook_Open. Эта процедура использует функцию Weekday для определения дня недели. Если текущий день недели — пятница, то отображается окно сообщения, которое напоминает пользователю о необходимости проведения еженедельного резервного копирования файлов. Если текущий день недели — не пятница, то ничего не происходит.

```
Private Sub Workbook_Open()
   If Weekday(Now) = vbFriday Then
      Msg = "Сегодня пятница. Ты уже создал"
      Msg = Msg & "новую резервную копию!"
      MsgBox Msg, vbInformation
   End If
End Sub
```

Событие Activate

Следующая процедура выполняется при каждой активизации рабочей книги. Эта процедура разворачивает активное окно.

```
Private Sub Workbook_Activate()
   ActiveWindow.WindowState = xlMaximized
End Sub
```

Событие SheetActivate

Представленная далее процедура выполняется каждый раз, когда пользователь активизирует один из листов рабочей книги. Если этот лист является рабочим, то выделяется ячейка A1. Если он не является рабочим листом, то ничего не происходит. Данная процедура использует функцию VBA TypeName для того, чтобы удостовериться, что активный лист является рабочим листом (не представлен листом диаграммы).

```
Private Sub Workbook_SheetActivate(ByVal Sh As Object)
   If TypeName(Sh) = "Worksheet" Then _
      Range("A1").Select
End Sub
```

Существует и другой метод избежать ошибки при выделении ячейки на листе диаграммы — просто игнорируйте такую ошибку.

```
Private Sub Workbook_SheetActivate(ByVal Sh As Object)
   On Error Resume Next
   Range("A1").Select
End Sub
```

Событие NewSheet

Рассматриваемая процедура выполняется каждый раз, когда к рабочей книге добавляется лист. В этом случае лист передается в процедуру в качестве аргумента. Поскольку новый лист может выступать как рабочим листом, так и листом диаграммы, эта процедура обязательно определяет тип листа. Если вставляется рабочий лист, то код создает запись о дате и времени добавления листа в ячейку A1.

```
Private Sub Workbook_NewSheet(ByVal Sh As Object)
   If TypeName(Sh) = "Worksheet" Then
      Sh.Cells.ColumnWidth = 35
      Sh.Range("A1") = "Лист добавлен " & Now()
   End If
End Sub
```

Событие BeforeSave

Событие BeforeSave возникает перед фактическим сохранением рабочей книги. Как известно, использование команды Файл⇨Сохранить (File⇨Save) приводит к появлению диалогового окна Сохранение документа (Save As). Это происходит в том случае, если рабочая книга еще ни разу не сохранялась и не открывалась в режиме "только чтение".

При выполнении процедуры Workbook_BeforeSave ей передается аргумент (SaveAsUI), который позволяет предвосхитить появление диалогового окна Сохранение документа. Следующий пример демонстрирует использование этого события.

```
Private Sub Workbook_BeforeSave _
   (ByVal SaveAsUI As Boolean, Cancel As Boolean)
   If SaveAsUI Then
      MsgBox "Убедитесь в том, что этот файл будет _
              сохранен на диске J."
   End If
End Sub
```

Когда пользователь сохраняет рабочую книгу, выполняется процедура Workbook_BeforeSave. Если операция сохранения приводит к появлению диалогового окна Сохранение документа, то переменной SaveAsUI присваивается значение True. Процедура проверяет переменную SaveAsUI, в результате чего сообщение отображается лишь в том случае, если на экране появляется окно сохранения документа. В случае присвоения процедурой аргументу Cancel значения True файл сохранен не будет.

Событие Deactivate

Предлагаемый пример демонстрирует использование события Deactivate. Эта процедура выполняется каждый раз при деактивизации (отмене выбора) рабочей книги. Она не позволяет пользователю деактивизировать рабочую книгу. При возникновении события Deactivate код выполняет повторную активизацию рабочей книги и выводит на экран соответствующее сообщение.

```
Private Sub Workbook_Deactivate()
  Me.Activate
  MsgBox "Мне жаль, но вы не покинете эту книгу..."
End Sub
```

Примечание

Пользоваться подобными процедурами, которые "берут на себя" функции Excel, не рекомендуется. Это может привести пользователя в замешательство. Пользователи должны уметь правильно работать с приложением.

Приведенный пример иллюстрирует важность обработки событий в необходимой последовательности. Запустив эту процедуру, вы убедитесь, что она работает правильно только в тех случаях, когда пользователь активизирует другую рабочую книгу. Событие Deactivate для рабочей книги возникает и в ряде других случаев:

- при закрытии рабочей книги;
- при открытии новой рабочей книги;
- при сворачивании окна рабочей книги.

Таким образом, данная процедура не будет работать точно так, как задумывалось изначально. Она будет предотвращать попытки пользователя непосредственно активизировать другую рабочую книгу, но не сможет препятствовать закрытию рабочей книги, открытию новой рабочей книги или сворачиванию окна рабочей книги. Окно сообщения будет отображаться, но действия все же будут возможны.

Событие BeforePrint

Событие BeforePrint происходит, когда пользователь запрашивает печать или предварительный просмотр печати, но фактической печати или предварительного просмотра не происходит. Событие использует аргумент Cancel, поэтому использующий это событие код может отменять печать или предварительный просмотр документа путем присвоения переменной Cancel значения True. К сожалению, определить, вызвано событие BeforePrint запросом на печать или запросом на предварительный просмотр, невозможно.

Обновление верхнего или нижнего колонтитула

Несмотря на то что параметры верхних и нижних колонтитулов Excel настраиваются в широких пределах, не существует способа печатать содержимое указанной ячейки верхнего или нижнего колонтитула непосредственно из Excel. С помощью события `Workbook_BeforePrint` можно отобразить содержимое текущей ячейки верхнего либо нижнего колонтитула в процессе печати рабочей книги. Приведенный ниже код обновляет левые нижние колонтитулы страниц при печати либо предварительном просмотре рабочей книги, а именно — вставляет содержимое ячейки A1 на лист `Лист1`.

```
Private Sub Workbook_BeforePrint(Cancel As Boolean)
   Dim sht As Object
   For Each sht In ThisWorkbook.Sheets
      sht.PageSetup.LeftFooter = _
         Worksheets("Лист1").Range("A1")
   Next sht
End Sub
```

Эта процедура циклически обходит каждый лист в рабочей книге, а также устанавливает свойство `LeftFooter` объекта `PageSetup` равным значению в ячейке A1 на листе `Лист1`.

Сокрытие столбцов перед печатью

В следующем примере используется процедура `Workbook_BeforePrint` для сокрытия столбцов B:D на листе `Лист1` перед печатью или предварительным просмотром.

```
Private Sub Workbook_BeforePrint(Cancel As Boolean)
'   Сокрытие столбцов B:D на листе Лист1 перед печатью
   Worksheets("Лист1").Range("B:D").EntireColumn.Hidden = True
End Sub
```

В идеальном случае следует отобразить скрытые ранее столбцы после завершения печати. Было бы неплохо, если бы в Excel существовало событие `AfterPrint`, выполняющее подобную операцию, но увы... об этом можно только мечтать. Не расстраивайтесь раньше времени: способ автоматического отображения ранее скрытых столбцов все же существует. Приведенная ниже процедура использует событие `OnTime`, с помощью которого вызывает процедуру `UnhideColumns` через пять секунд после завершения печати или предварительного просмотра.

```
Private Sub Workbook_BeforePrint(Cancel As Boolean)
'  Скрывает столбцы B:D на листе Лист1 перед печатью
   Worksheets("Лист1").Range("B:D").EntireColumn.Hidden = True
   Application.OnTime Now + TimeValue("0:00:05"), "UnhideColumns"
End Sub
```

Процедура `UnhideColumns` находится в стандартном модуле VBA.

```
Sub UnhideColumns()
   Worksheets("Лист1").Range("B:D").EntireColumn.Hidden = False
End Sub
```

Компакт-диск

Этот пример находится на прилагаемом к книге компакт-диске в файле `hide columns before printing.xlsm`.

Перекрестная ссылка

Для получения дополнительных сведений о событиях `OnTime` обратитесь к разделу "Событие `OnTime`".

Событие BeforeClose

Событие BeforeClose происходит перед закрытием рабочей книги. Оно часто используется совместно с процедурой Workbook_Open. Например, можно обратиться к процедуре Workbook_Open для создания дополнительной опции меню, специфичной для конкретной рабочей книги, а процедуру Workbook_BeforeClose применить для удаления этого дополнительного элемента из меню. Таким образом, вы обеспечите доступность опции меню только в открытой рабочей книге.

К сожалению, событие Workbook_BeforeClose реализовано недостаточно хорошо. Например, при попытке закрытия несохраненной рабочей книги Excel отображает запрос, в котором пользователю предлагается сохранить рабочую книгу перед ее закрытием, как показано на рис. 19.5. Проблема заключается в том, что к моменту появления этого сообщения событие Workbook_BeforeClose уже произошло. И если пользователь не захочет сохранять рабочую книгу, процедура обработчика событий завершит свою работу.

Рис. 19.5. К моменту появления этого сообщения процедура Workbook_BeforeClose уже завершит свою работу

Рассмотрим иной сценарий развития событий. В момент открытия определенной рабочей книги отображается пользовательское меню. В данном случае рабочая книга использует процедуру Workbook_Open для создания меню при открытии рабочей книги, а процедура Workbook_BeforeClose — для удаления этого меню при закрытии рабочей книги. Процедуры обработки событий приведены ниже. Обе процедуры вызывают другие процедуры, которые здесь не рассматриваются.

```
Private Sub Workbook_Open()
   Call CreateShortcutMenuItems
End Sub

Private Sub Workbook_BeforeClose(Cancel As Boolean)
   Call DeleteShortcutMenuItems
End Sub
```

Как отмечалось выше, сообщение Excel Сохранить изменения... отображается после того, как процедура Workbook_BeforeClose завершит свою работу. Поэтому если пользователь щелкнет на кнопке Отмена, то рабочая книга останется открытой, но дополнительный элемент меню будет уже удален!

Одним из решений подобной проблемы является создание кода запроса на сохранение в процедуре Workbook_BeforeClose. Ниже приведен пример, демонстрирующий способ выполнения подобной задачи.

```
Private Sub Workbook_BeforeClose(Cancel As Boolean)
   Dim Msg As String
   If Me.Saved = False Then
      Msg = "Сохранить изменения в документе? "
      Msg = Msg & Me.Name & "?"
      Ans = MsgBox(Msg, vbQuestion + vbYesNoCancel)
      Select Case Ans
```

```
        Case vbYes
           Me.Save
        Case vbCancel
           Cancel = True
           Exit Sub
        End Select
     End If
     Call DeleteShortcutMenuItems
     Me.Saved = True
End Sub
```

Данная процедура проверяет значение свойства `Saved` объекта `Workbook` с целью определить наличие в рабочей книге несохраненных данных. Если они существуют, то выполняется процедура `DeleteShortcutMenuItems`, и рабочая книга закрывается. Однако в рабочей книге могут оставаться несохраненные данные. Тогда процедура отображает окно сообщения, которое дублирует исходное окно сообщения Excel. В этом окне находятся следующие три кнопки.

- **Да**. Рабочая книга сохраняется, элемент меню удаляется, а рабочая книга закрывается.
- **Нет**. Этот код присваивает свойству `Saved` объекта `Workbook` значение `True` (при этом файл фактически не сохраняется), удаляет элемент меню и закрывает файл.
- **Отмена**. Событие `BeforeClose` отменяется, и процедура завершается без удаления элементов контекстного меню.

Компакт-диск

Рабочая книга, содержащая этот пример, находится на прилагаемом компакт-диске в файле `workbook_beforeclose workaround.xlsm`.

События объекта `Worksheet`

События объекта `Worksheet` — наиболее полезные и часто используемые. Контроль над ними может заставить приложение выполнять функции, которые в другой ситуации считались бы невозможными.

В табл. 19.2 перечислены события рабочего листа, сопровождаемые краткими описаниями.

Таблица 19.2. События объекта `Worksheet`

Событие	Действие, приводящее к возникновению события
`Activate`	Активизация рабочего листа
`BeforeDoubleClick`	Двойной щелчок на рабочем листе
`BeforeRightClick`	Щелчок правой кнопкой мыши на рабочем листе
`Calculate`	Вычисление (или повторное вычисление) значений рабочего листа
`Change`	Изменение пользователем или внешней ссылкой значения ячейки рабочего листа
`Deactivate`	Деактивизация рабочего листа
`FollowHyperlink`	Щелчок на гиперссылке в рабочем листе
`PivotTableUpdate`	Обновление сводной таблицы на рабочем листе
`SelectionChange`	Изменение либо обновление выделенной области на рабочем листе

Помните, что код процедуры обработки события рабочего листа должен храниться в модуле кода соответствующего объекта рабочего листа.

Совет

Для быстрого перехода к окну модуля кода щелкните правой кнопкой мыши на ярлычке листа и выберите пункт Исходный текст (View Code).

Событие Change

Событие Change возникает в момент изменения значения ячейки пользователем или VBA-процедурой. Событие Change не возникает, когда расчеты приводят к появлению другого значения формулы или когда на рабочий лист добавляется новый объект.

При вызове процедуры Worksheet_Change в качестве аргумента Target ей передается объект Range. Этот объект представляет изменившуюся ячейку (или диапазон), которая привела к возникновению события. В следующем примере отображается окно сообщения, которое выводит адрес диапазона, указанного в параметре Target.

```
Private Sub Worksheet_Change(ByVal Target As Excel.Range)
  MsgBox "Диапазон " & Target.Address & " изменился."
End Sub
```

Для того чтобы получить представление о типах действий, которые приводят к возникновению события Change в рабочем листе, добавьте предыдущую процедуру в модуль кода объекта Worksheet. После ввода этой процедуры активизируйте Excel и внесите изменения в рабочий лист, используя для этого различные методы. Каждый раз при возникновении события Change будет отображаться адрес диапазона, который изменился.

После запуска этой процедуры вами может быть замечена интересная особенность: некоторые действия, которые должны способствовать возникновению этого события, ни к чему не приводят, а те, которые не должны выполнять эту задачу, приводят к возникновению события Change!

- Изменение форматирования ячейки не приводит к возникновению события Change, а использование команды Главная⇨Редактирование⇨Очистить⇨Очистить форматы (Home⇨Editing⇨Clear⇨Clear Formats), наоборот, вызывает это событие. Также это событие вызывается при использовании диалогового окна Специальная вставка (Paste Special).
- Добавление, редактирование или удаление комментария в ячейке не приводит к возникновению события Change.
- Нажатие клавиши <Del> приводит к генерации события Change даже в том случае, если в ячейке не содержится данных.
- Ячейки, которые изменяются с помощью команд Excel, могут генерировать, а могут и не генерировать событие Change. Например, команды сортировки диапазона не генерируют это событие, а проверка грамматики — генерирует.
- Если процедура VBA изменяет содержимое ячейки, то это приводит к возникновению события Change.

Рассматривая приведенный выше список, вы убедитесь, что использовать событие Change для отслеживания изменений в ячейках нецелесообразно.

Отслеживание изменений в определенном диапазоне

Событие Change возникает при внесении изменений в любую ячейку рабочего листа. Но в большинстве случаев важно отслеживать изменения, которые вносятся в определенную ячейку или диапазон. Когда вызывается процедура обработки события Worksheet_Change, она получает в качестве параметра объект Range. Этот объект представляет диапазон ячеек или ячейку, которая после изменения приводит к возникновению события Change.

Предположим, что на рабочем листе определен диапазон InputRange, и вам необходимо отслеживать только те изменения, которые внесены в этом диапазоне. Не существует события Change для отдельного объекта Range, но можно выполнить необходимую проверку в начале процедуры Worksheet_Change.

```
Private Sub Worksheet_Change(ByVal Target As Excel.Range)
   Dim MRange As Range
   Set MRange = Range("InputRange")
   If Not Intersect(Target, MRange) Is Nothing Then _
      MsgBox "Изменена ячейка текущего диапазона."
End Sub
```

В данном примере используется объект Range, который называется VRange. Он представляет диапазон ячеек на рабочем листе, который необходимо проверять на предмет внесения изменений. Процедура использует функцию VBA Intersect, чтобы определить расположение диапазона Target (полученного в качестве атрибута процедуры) в диапазоне VRange. Функция Intersect возвращает объект, который состоит из всех ячеек, содержащихся в обоих аргументах. Если функция Intersect возвращает значение Nothing, то эти диапазоны не имеют общих ячеек. Оператор отрицания Not используется для того, чтобы выражение стало равным True в том случае, если указанный диапазон имеет хотя бы одну общую ячейку с диапазоном VRange. Таким образом, будет отображено окно сообщения. В противном случае ничего не происходит, и процедура завершает свою работу.

Выделение формул полужирным шрифтом

В следующем примере выделяются полужирным шрифтом все формулы рабочего листа.

```
Private Sub Worksheet_Change(ByVal Target As Excel.Range)
   Dim cell As Range
   For Each cell In Target
      cell.Font.Bold = cell.HasFormula
   Next cell
End Sub
```

Поскольку передаваемый процедуре Worksheet_Change объект может включать диапазон, состоящий из ряда ячеек, процедура выполняет цикл по всем ячейкам в диапазоне Target. Если ячейка содержит формулу, эта формула выделяется полужирным шрифтом. Если же формулы в ячейке нет, свойству Bold присваивается значение False.

Эта процедура работает, хотя и неидеально. Что произойдет, если пользователь удалит строку или столбец? В подобном случае диапазон Target будет состоять из большого количества ячеек. Проверка всех этих ячеек с помощью цикла For Each отнимет очень много времени, и в итоге требуемые формулы не будут найдены.

В приведенной ниже модифицированной процедуре эта проблема решается путем изменения диапазона Target таким образом, чтобы он представлял собой результат пересечения диапазона Target и диапазона, используемого рабочим листом. Для того чтобы

не допустить возникновение ситуации, когда Target равен Nothing, исключается случай, когда пустая строка или столбец находится за пределами используемого диапазона.

```
Private Sub Worksheet_Change(ByVal Target As Excel.Range)
   Dim cell As Range
   Set Target = Intersect(Target, Target.Parent.UsedRange)
   If Not Target Is Nothing Then
      For Each cell In Target
         cell.Font.Bold = cell.HasFormula
      Next cell
   End If
End Sub
```

Компакт-диск

Этот пример находится на прилагаемом к книге компакт-диске в файле make formulas bold.xlsm.

Предупреждение

Использование процедур Worksheet_Change ведет к отключению свойства отмены операций Excel. При вызове подобного макроса происходит разрушение стека отмены операций. При использовании процедуры обработки события Worksheet_Change вызывается макрос при каждом изменении рабочего листа.

Проверка правильности введенных данных

Средство Excel проверки данных может оказаться очень ценным инструментом, но его применение может вызвать следующую серьезную проблему. При вставке данных в ту ячейку, в которой реализуется проверка данных, не только не проверяется значение, но и правила проверки, которые связаны с этой ячейкой, безвозвратно удаляются! Таким образом, инструмент проверки данных становится практически бесполезным в собственных приложениях. В этом разделе продемонстрированы методы использования события Change объекта Worksheet для создания процедур проверки правильности введенных данных.

Компакт-диск

На прилагаемом к книге компакт-диске находятся две версии этого примера. В первом примере (под названием validate entry1.xlsm) используется свойство EnableEvents для предотвращения бесконечного цикла возникновения событий Change; во втором (под названием validate entry2.xlsm) используется статическая переменная (см. раздел "Отключение событий").

Процедура Worksheet_Change, код которой приведен ниже, вызывается в случае, когда пользователь изменяет ячейку. Проверка ограничена диапазоном InputRange. В этот диапазон разрешается вводить только целые значения от 1 до 12.

```
Private Sub Worksheet_Change(ByVal Target As Range)
   Dim VRange As Range, cell As Range
   Dim Msg As String
   Dim ValidateCode As Variant
   Set VRange = Range("InputRange")
   If Intersect(VRange, Target) Is Nothing Then Exit Sub
   For Each cell In Intersect(VRange, Target)
      ValidateCode = EntryIsValid(cell)
```

```
      If TypeName(ValidateCode) = "String" Then
         Msg = "Ячейка " & cell.Address(False, False) & ":"
         Msg = Msg & vbCrLf & vbCrLf & ValidateCode
         MsgBox Msg, vbCritical, "Некорректная запись"
         Application.EnableEvents = False
         cell.ClearContents
         cell.Activate
         Application.EnableEvents = True
      End If
   Next cell
End Sub
```

Процедура `Worksheet_Change` создает объект `Range`. Он называется `VRange` и представляет проверяемый диапазон на рабочем листе. В результате циклически просматриваются все ячейки аргумента `Target`, представляющего диапазон изменившихся ячеек. В коде определяется, находится ли изменившаяся ячейка в проверяемом диапазоне, и если это так, передает ячейку в качестве аргумента другой функции (`EntryIsValid`), возвращающей значение `True` для действительного значения ячейки.

Если значение ячейки выходит за пределы набора допустимых значений, то функция `EntryIsValid` возвращает строку, которая описывает проблему. Затем выводится окно сообщения (рис. 19.6). После закрытия окна сообщения неправильное значение ячейки удаляется, и ячейка активизируется. Обратите внимание, что перед удалением значения ячейки события отключаются. Если события не отключить, то ячейка создаст событие `Change` и введет процедуру в бесконечный цикл.

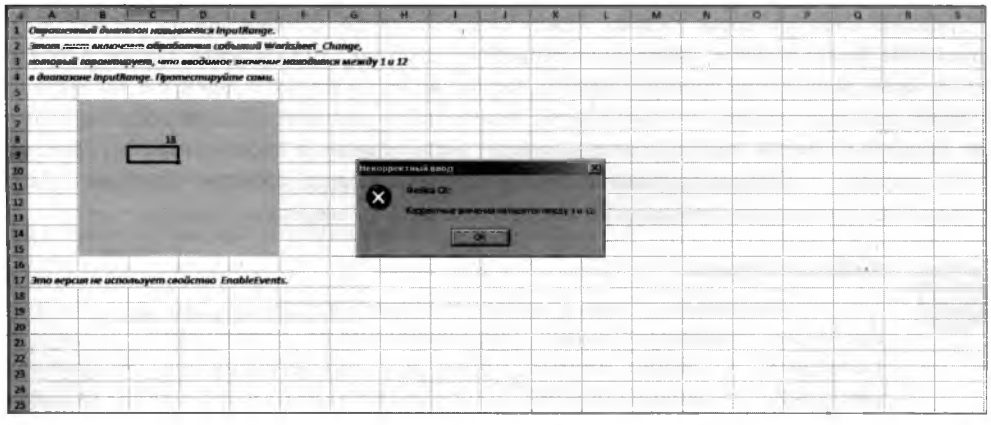


Рис. 19.6. В этом окне сообщения описывается проблема, возникающая при вводе пользователем неверных данных

Ниже приводится код функции `EntryIsValid`.

```
Private Function EntryIsValid(cell) As Variant
'  Возвращает True, если в ячейку вводится целое число в
'  диапазоне от ' 1 до 12.
'  В противном случае возвращается строка, описывающая проблему.

'  Число?
   If Not WorksheetFunction.IsNumber (cell) Then
      EntryIsValid = "Нечисловое значение."
      Exit Function
   End If
```

```
' Целое?
   If CInt(cell) <> cell Then
      EntryIsValid = "Введите целое число."
      Exit Function
   End If

' Диапазон?
   If cell < 1 Or cell > 12 Then
      EntryIsValid = "Значение должно находиться в диапазоне _
         между 1 и 12."
      Exit Function
   End If
' Тест завершен успешно
   EntryIsValid = True
End Function
```

Описанная методика вполне пригодна для применения на практике, хотя и сложна в настройке. Было бы неплохо использовать свойство проверки правильности введенных данных Excel и гарантировать при этом сохранение правил, тестирующих корректность вводимых данных, при их вставке в проверяемый диапазон. Для этого воспользуйтесь следующим примером кода.

```
Private Sub Worksheet_Change(ByVal Target As Range)
   Dim VT As Long
' Все ли ячейки в диапазоне
' прошли процедуру проверки?
   On Error Resume Next
   VT = Range("InputRange").Validation.Type
   If Err.Number <> 0 Then
      Application.Undo
      MsgBox "Отменена последняя операция." & _
             "Удалены правила проверки данных.", vbCritical
   End If
End Sub
```

Эта процедура обработки событий проверяет статус проверки данных диапазона (под именем `InputRange`), в котором *предположительно* содержатся правила проверки вводимых данных. Если переменная VT содержит ошибку, значит, одна или более ячеек в диапазоне `InputRange` больше не включают правил проверки вводимых данных. Подобное явление может быть результатом копирования данных в диапазон, который содержит правила проверки вводимых данных. В этом случае код вызывает метод `Undo` объекта `Application`, отменяя тем самым действие, выполненное пользователем. При этом отображается окно сообщения, показанное на рис. 19.7.

Примечание

Благодаря использованию этой процедуры предотвращается разрушение стека отмены операций.

Компакт-диск

Рассматриваемый в этом разделе пример находится на прилагаемом к книге компакт-диске в файле `validate entry3.xlsm`.

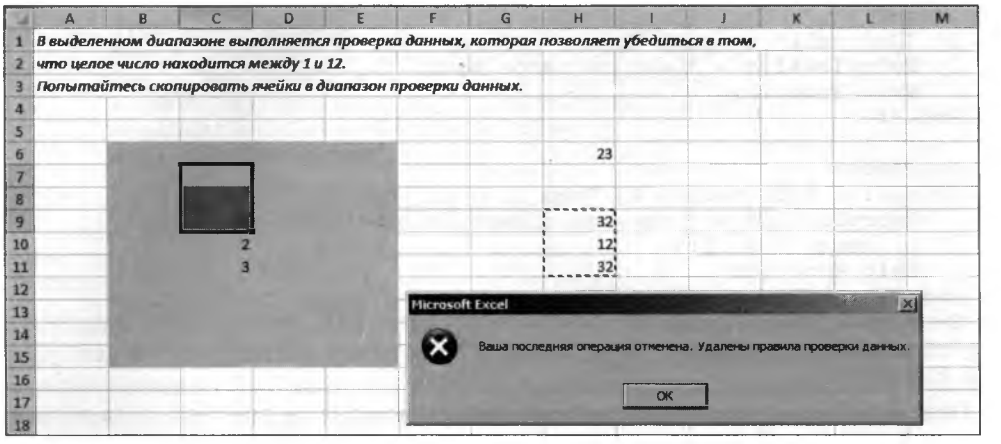


Рис. 19.7. Процедура `Worksheet_Change` позволяет зафиксировать факт удаления правила проверки вводимых данных

Событие `SelectionChange`

Следующая процедура демонстрирует использование события `SelectionChange`. Она выполняется каждый раз, когда пользователь создает новое выделение на рабочем листе.

```
Private Sub Worksheet_SelectionChange(ByVal Target As Excel.Range)
    Cells.Interior.ColorIndex = xlNone
    With ActiveCell
        .EntireRow.Interior.Color = RGB(219, 229, 241)
        .EntireColumn.Interior.Color = RGB(219, 229, 241)
    End With
End Sub
```

Данная процедура выделяет строку и столбец активной ячейки, что облегчает визуальный поиск последней. Первый оператор нейтрализует цвет фона для всех ячеек рабочего листа, после чего строка и столбец активной ячейки выделяются светло-голубым цветом. На рис. 19.8 отображен эффект выделения (поверьте, это голубой цвет).

Если рабочий лист окрашен в фоновый цвет, эта процедура обычно не применяется. Исключения из этого правила — таблицы с выбранным стилем, а также рабочие листы, которые были окрашены в фоновый цвет в результате применения фонового форматирования. Учтите, что в результате выполнения макроса `Worksheet_SelectionChange` разрушается стек отмены операций, поэтому использование подобной техники приведет к отключению свойства отмены операций Excel.

Компакт-диск

Этот пример находится на прилагаемом к книге компакт-диске в файле `shade active row and column.xlsm`.

Событие `BeforeDoubleClick`

Можно создать процедуру VBA, которая вызывается на выполнение после двойного щелчка мышью на ячейке. В следующем примере (код этого примера хранится в окне кода объекта `Sheet`) двойной щелчок на ячейке приводит к ее выделению полужирным

шрифтом (стиль "Good", если она подобным шрифтом не выделена) либо к отмене подобного выделения (стиль "Normal").

```
Private Sub Worksheet_BeforeDoubleClick _
   (ByVal Target As Excel.Range, Cancel As Boolean)
   If Target.Style = "Good" Then
      Target.Style = "Normal"
   Else
      Target.Style = "Good"
   End If
   Cancel = True
End Sub
```

	A	B	C	D	E	F	G
1		Проект-1	Проект-2	Проект-3	Проект-4	Проект-5	Проект-6
2	янв-2007	2158	1527	3870	4863	3927	39
3	фев-2007	4254	28	4345	2108	412	28
4	мар-2007	3631	1240	4208	452	3443	29
5	апр-2007	724	4939	1619	1721	3631	34
6	май-2007	3060	1034	1646	345	978	5
7	июн-2007	394	1241	2965	1411	3545	44
8	июл-2007	2080	3978	3304	1460	4533	33
9	авг-2007	411	753	732	1207	1902	40
10	сен-2007	2711	95	2267	2634	1944	39
11	окт-2007	2996	4934	3932	2938	4730	11
12	ноя-2007	2837	1116	3879	1740	1466	36
13	дек-2007	300	2917	321	1219	841	35
14	янв-2008	1604	768	2617	3414	4732	8
15	фев-2008	1662	1380	4590	531	4143	17
16	мар-2008	1001	3454	4611	4852	456	
17	апр-2008	4407	46	4185	4868	2313	27
18	май-2008	3948	1292	1462	1977	2418	18
19	июн-2008	160	2908	3834	2396	4120	22

Sheet1

Рис. 19.8. Перемещение курсора ячейки приводит к окрашиванию активной строки и столбца

Если аргументу `Cancel` присвоено значение `True`, блокируется операция, выполняемая по умолчанию после двойного щелчка мышью. Другими словами, двойной щелчок мышью на ячейке не приводит к переключению Excel в режим редактирования ячейки.

Событие `BeforeRightClick`

Когда пользователь щелкает правой кнопкой мыши на рабочем листе, Excel отображает контекстное меню. Если по какой-либо причине необходимо отключить появление контекстного меню на одном из рабочих листов, то можно перехватить событие `BeforeRightClick`. Следующая процедура устанавливает значение аргумента `Cancel` равным `True`, что приводит к отмене обработки события `BeforeRightClick`. Об этом говорит появляющееся на экране сообщение.

```
Private Sub Worksheet_BeforeRightClick _
   (ByVal Target As Excel.Range, Cancel As Boolean)
   Cancel = True
   MsgBox "Контекстное меню недоступно."
End Sub
```

Обратите внимание, что пользователь по-прежнему может получить доступ к контекстному меню, нажав комбинацию клавиш <Shift+F10>. Но далеко не все пользователи Excel об этом знают.

Перекрестная ссылка

Если вы интересуетесь способом перехвата комбинации клавиш <Shift+F10>, обратитесь к разделу "Событие `OnKey`". В главе 23 описываются другие методы отключения контекстных меню.

Ниже представлен еще один пример, демонстрирующий использование события `BeforeRightClick`. Эта процедура проверяет наличие числового значения в ячейке, на которой выполнен щелчок правой кнопкой мыши. Если числовое значение обнаружено, отображается диалоговое окно **Формат ячеек** (Format Cells), открытое на вкладке **Число** (Number), а аргументу `Cancel` присваивается значение `True` (при этом стандартное контекстное меню не отображается). Если в ячейке не содержится числовое значение, ничего не происходит — отображается стандартное контекстное меню.

```
Private Sub Worksheet_BeforeRightClick _
    (ByVal Target As Excel.Range, Cancel As Boolean)
    If IsNumeric(Target) And Not IsEmpty(Target) Then
       Application.Dialogs(xlDialogFormatNumber).Show
       Cancel = True
    End If
End Sub
```

Обратите внимание, что код осуществляет дополнительную проверку ячейки, чтобы убедиться в ее "непустоте". Необходимость такой проверки обусловлена тем, что VBA рассматривает пустые ячейки как содержащие числовые значения.

События объекта `Chart`

По умолчанию события разрешены только для диаграмм, которые находятся на листах диаграмм. Для того чтобы работать с событиями встроенных диаграмм, необходимо создать модуль класса.

Перекрестная ссылка

Дополнительные примеры управления событиями объекта `Chart` можно найти в главе 18. Здесь же приведено описание методов создания модуля класса, применяемого для управления событиями встроенных диаграмм.

В табл. 19.3 представлены список событий, поддерживаемых диаграммами, а также краткое описание каждого события.

Таблица 19.3. События, поддерживаемые листами диаграмм

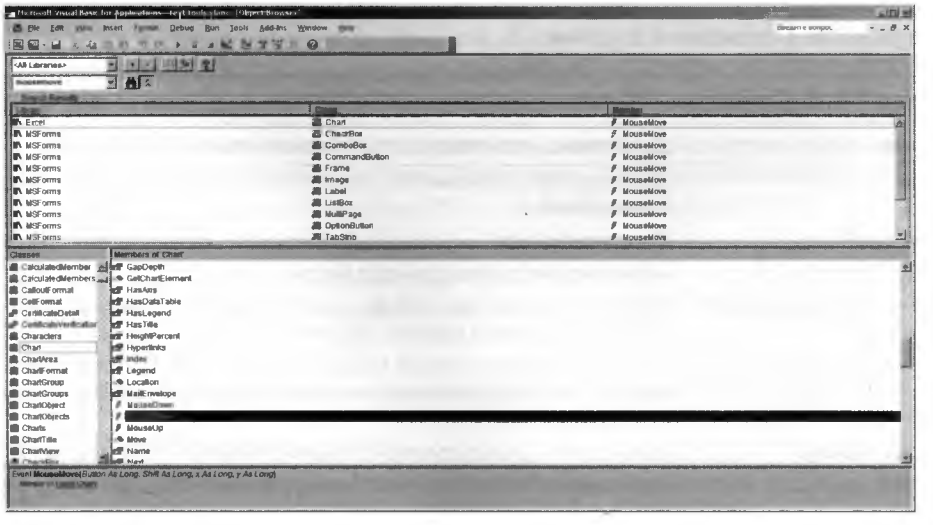
Событие	Действие, приводящее к возникновению события
`Activate`	Активизация листа диаграммы или встроенной диаграммы
`BeforeDoubleClick`	Двойной щелчок мышью на листе диаграммы или встроенной диаграмме (событие происходит до обработки двойного щелчка, принятого по умолчанию)

Окончание табл. 19.3

Событие	Действие, приводящее к возникновению события
`BeforeRightClick`	Щелчок правой кнопкой мыши на листе диаграммы или на встроенной диаграмме (событие происходит до обработки такого щелчка мыши, принятого по умолчанию)
`Calculate`	Отображение на диаграмме новых или измененных данных
`Deactivate`	Отмена выделения диаграммы
`MouseDown`	Нажатие кнопки мыши в момент, когда указатель находится над диаграммой
`MouseMove`	Перемещение указателя мыши над диаграммой
`MouseUp`	Отпускание кнопки мыши в момент, когда указатель находится над диаграммой
`Resize`	Изменение размеров диаграммы
`Select`	Выделение элемента диаграммы
`SeriesChange`	Изменение значения точки данных в одном из рядов диаграммы

Использование браузера объектов для поиска событий

Окно Object Browser (браузер объектов) является весьма полезным инструментом, поскольку позволяет получить информацию об объектах, а также их свойствах и методах. Также оно позволяет определить объекты, поддерживающие определенное событие. Например, предположим, что требуется найти объекты, которые поддерживают событие `MouseMove`. Для этого активизируйте редактор VBE и нажмите клавишу <F2> для отображения окна Object Browser. Удостоверьтесь в том, что выбран пункт <All Libraries> (<Все библиотеки>), после чего введите `MouseMove` и щелкните на значке с изображением бинокля (см. показанный ниже рисунок).



В окне Object Browser отображается список подходящих объектов. События обозначаются с помощью небольшой желтой молнии. Используя этот список, можно определить, какие объекты поддерживают событие `MouseMove`. Большинство из найденных объектов представляют собой элементы управления, находящиеся в библиотеке MSForms, вклю-

чающей элемент управления UserForm. Это событие также поддерживается объектом Excel Chart.

Обратите внимание на то, что список делится на три столбца: Library (Библиотека), Class (Класс) и Member (Элемент). Элемент, соответствующий критерию поиска, может оказаться в любом из этих столбцов. Это приводит к сложной проблеме, связанной с тем, что имя события или термина относится к двум библиотекам или классам, хотя не всегда представляет одинаковую функциональность. Поэтому следует щелкнуть на каждом из интересующих элементов, отображенных в окне Object Browser, и сравнить информацию, которая приводится под списком. Может оказаться, что отдельный класс или библиотека рассматривает события различным образом.

События объекта Application

В предыдущих разделах были описаны события объектов Workbook и Worksheet. Эти события рассматривались по отношению к определенной рабочей книге или рабочему листу. Если требуется проконтролировать возникновение событий для всех открытых рабочих книг или рабочих листов, рекомендуется обратиться к событиям уровня объекта Application.

Примечание

Создание процедуры обработки события для объекта Application требует создания модуля класса и выполнения дополнительной работы по настройке среды.

В табл. 19.4 перечислены события объекта Application с кратким описанием каждого из них. В Excel 2010 появилось несколько новых событий, имеющих отношение к защищенному режиму окон и сводным таблицам. Дополнительные сведения по этой теме можно найти в справочной системе.

Таблица 19.4. События, поддерживаемые объектом Application

Событие	Действие, приводящее к возникновению события
AfterCalculate	Завершение вычислений, отсутствие очереди на выполнение вычислений
NewWorkbook	Создание книги
SheetActivate	Активизация произвольного листа
SheetBeforeDoubleClick	Двойной щелчок на любом листе (событие происходит до обработки двойного щелчка, принятого по умолчанию)
SheetBeforeRightClick	Щелчок правой кнопкой мыши на любом листе (событие происходит до обработки щелчка правой кнопкой мыши, принятого по умолчанию)
SheetCalculate	Вычисление (либо повторное вычисление) любого листа
SheetChange	Изменение пользователем либо внешней ссылкой ячеек в любом листе
SheetDeactivate	Деактивизация любого листа
SheetFollowHyperlink	Щелчок на гиперссылке
SheetPivotTableUpdate	Обновление любой сводной таблицы
SheetSelectionChange	Изменение выделения на любом листе, кроме листа диаграммы

Окончание табл. 19.4

Событие	Действие, приводящее к возникновению события
WindowActivate	Активизация окна любой рабочей книги
WindowDeactivate	Отменена выделения окна любой рабочей книги
WindowResize	Изменение размеров окна любой рабочей книги
WorkbookActivate	Активизация любой рабочей книги
WorkbookAddinInstall	Установка рабочей книги в качестве надстройки
WorkbookAddinUninstall	Деинсталяция любой надстройки в виде рабочей книги
WorkbookBeforeClose	Закрытие любой рабочей книги
WorkbookBeforePrint	Печать любой открытой рабочей книги
WorkbookBeforeSave	Сохранение любой открытой рабочей книги
WorkbookDeactivate	Деактивизация любой открытой рабочей книги
WorkbookNewSheet	Создание листа в любой открытой рабочей книге
WorkbookOpen	Открытие рабочей книги

Включение событий уровня объекта Application

Для использования событий уровня объекта Application выполните следующие действия.

1. Создайте модуль класса.
2. В окне Properties (Свойства) присвойте имя этому модулю класса.

   По умолчанию VBA присваивает каждому новому модулю класса заданные по умолчанию имена Класс1, Класс2 и т.д. Можно присвоить модулю класса другое имя, которое лучше описывает его назначение, например clsApp.
3. В модуле класса объявите глобальный объект Application с помощью ключевого слова WithEvents, как показано ниже.

   ```
   Public WithEvents XL As Apolication
   ```
4. Создайте переменную, которая будет использоваться в качестве ссылки на объект Application в модуле класса. Это должна быть переменная уровня модуля, объявленная в обычном модуле VBA (а не в модуле класса).

   ```
   Dim X As New clsApp
   ```
5. Свяжите объявленную переменную с объектом Application. Обычно эта операция выполняется в процедуре Workbook_Open, как показано ниже.

   ```
   Set X.XL = Application
   ```
6. Создайте процедуры-обработчики событий для объекта XL в модуле класса.

Перекрестная ссылка

Эти действия практически идентичны выполняемым при обработке событий на уровне объекта встроенной диаграммы (см. главу 18).

Определение факта открытия рабочей книги

Пример, приведенный в этом разделе, позволяет отслеживать события открытия каждой рабочей книги и сохранения информации в текстовом файле (в формате CSV).

Начать создание такой процедуры можно со вставки нового модуля класса, называющегося clsApp. В модуле класса должен располагаться следующий код.

```
Public WithEvents AppEvents As Application

Private Sub AppEvents_WorkbookOpen _
   (ByVal Wb As Excel.Workbook)
   Call UpdateLogFile(Wb)
End Sub
```

Этот код объявляет объект Application, который называется AppEvents и поддерживает обработку событий. Процедура AppEvents_WorkbookOpen вызывается каждый раз при открытии рабочей книги. Эта процедура обработки события вызывает процедуру UpdateLogFile и передает ей аргумент Wb, который представляет открытую рабочую книгу. После этого необходимо добавить модуль VBA, содержащий следующий код.

```
Dim AppObject As New clsApp

Sub Init()
'  Вызывается процедурой Workbook_Open
   Set AppObject.AppEvents = Application
End Sub

Sub UpdateLogFile(Wb)
   Dim txt As String
   Dim Fname As String
   txt = Wb.FullName
   txt = txt & "," & Date & "," & Time
   txt = txt & "," & Application.UserName
   Fname = Application.DefaultFilePath & "\logfile.csv"
   Open Fname For Append As #1
   Write #1, txt
   Close #1
   MsgBox txt
End Sub
```

Обратите внимание на объявление переменной типа AppClass (так называется модуль класса). Вызов процедуры Init производится в процедуре Workbook_Open, которая расположена в модуле кода объекта ЭтаКнига. Процедура Workbook_Open выглядит следующим образом.

```
Private Sub Workbook_Open()
   Call Init
End Sub
```

Процедура UpdateLogFile открывает текстовый файл (или создает его, если такого файла не существует) и записывает ключевую информацию об открытой рабочей книге: имя файла, полный путь к нему, дату, время и имя пользователя.

Процедура Workbook_Open вызывает процедуру Init. Таким образом, при открытии рабочей книги процедура Init создает новый объект.

Компакт-диск

Рассматриваемый в этом разделе пример находится на прилагаемом к книге компакт-диске в файле log workbook open.xlsm.

Отслеживание событий уровня объекта Application

Для того чтобы "прочувствовать" процесс генерации событий, воспользуйтесь списком событий, которые генерируются в процессе повседневной работы.

Я создал приложение, которое отображает (в окне UserForm) каждое событие, происходящее на уровне объекта Application (рис. 19.9). Этот пример может быть полезным при изучении типов и последовательности происходящих событий.

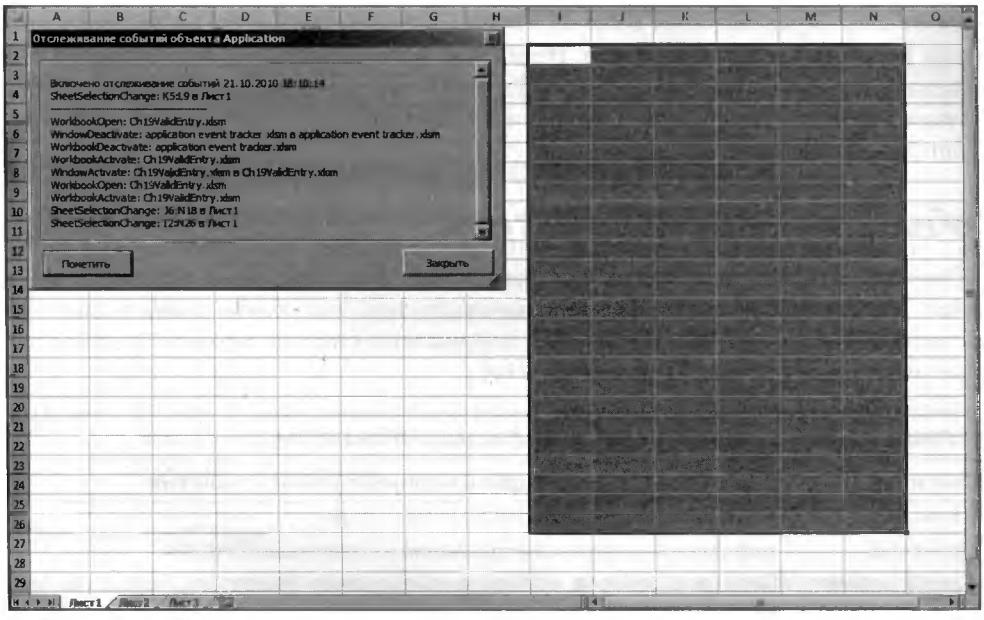


Рис. 19.9. В этой рабочей книге используется модуль класса для отслеживания событий, происходящих на уровне объекта Application

Компакт-диск

Рассматриваемый в этом разделе пример можно найти на прилагаемом к книге компакт-диске в файле application event tracker.xlsm.

В рабочей книге содержится модуль с 21 процедурой (по одной на каждое событие, поддерживаемое объектом Application). Приведем пример такой процедуры.

```
Private Sub XL_NewWorkbook(ByVal Wb As Excel.Workbook)
   LogEvent "NewWorkbook: " & Wb.Name
End Sub
```

Каждая процедура вызывает процедуру LogEvent и передает ей в качестве аргумента имя события и объект. Ниже приведен код процедуры LogEvent.

```
Sub LogEvent(txt)
   EventNum = EventNum + 1
   With UserForm1
      With .lblEvents
         .AutoSize = False
         .Caption = .Caption & vbCrLf & txt
         .Width = UserForm1.FrameEvents.Width - 20
         .AutoSize = True
```

```
        End With
        .FrameEvents.ScrollHeight = .lblEvents.Height + 20
        .FrameEvents.ScrollTop = EventNum * 20
    End With
End Sub
```

Процедура `LogEvent` обновляет диалоговое окно `UserForm`, изменяя свойство `Caption` элемента управления `Label`, который называется `lblEvents`. Кроме того, процедура изменяет значения свойств `ScrollHeight` и `ScrollTop` элемента управления `Frame`, называющегося `FrameEvents`. Элемент управления `Label` расположен внутри элемента управления `Frame`. Изменение значений этих свойств приведет к отображению недавно добавленного текста и к сокрытию ранее добавленного текста. Можно также изменить размер окна `UserForm` по вертикали. Для этого используется техника, описанная в главе 15.

События объекта `UserForm`

Объект `UserForm` поддерживает достаточно большое количество событий, а каждый элемент управления, размещенный в пользовательском диалоговом окне, поддерживает собственный набор событий. В табл. 19.5 перечислены события объекта `UserForm`, которые можно перехватывать с помощью VBA.

Таблица 19.5. События, поддерживаемые объектом `UserForm`

Событие	Действие, приводящее к возникновению события
`Activate`	Активизация диалогового окна `UserForm`
`AddControl`	Добавление элемента управления на этапе выполнения
`BeforeDragOver`	Перетаскивание в процессе выполнения (указатель мыши находится над диалоговым окном)
`BeforeDropOr-Paste`	Удаление или вставка данных пользователем (момент непосредственного отпускания кнопки мыши)
`Click`	Щелчок мышью в момент, когда ее указатель находится над диалоговым окном
`DblClick`	Двойной щелчок мышью, когда ее указатель находится над диалоговым окном
`Deactivate`	Деактивизация диалогового окна `UserForm`
`Error`	Вызов ошибки элементом управления, но отсутствие возможности передать сведения о ней вызывающей программе
`Initialize`	Загрузка в память диалогового окна `UserForm`
`KeyDown`	Нажатие клавиши
`KeyPress`	Нажатие пользователем любой клавиши, приводящей к вводу символа
`KeyUp`	Отпускание клавиши
`Layout`	Изменение размеров диалогового окна `UserForm`
`MouseDown`	Нажатие кнопки мыши
`MouseMove`	Перемещение указателя мыши
`MouseUp`	Отпускание кнопки мыши
`QueryClose`	Происходит перед закрытием диалогового окна `UserForm`
`RemoveControl`	Удаление элемента управления из диалогового окна на этапе выполнения

Окончание табл. 19.5

Событие	Действие, приводящее к возникновению события
`Resize`	Изменение размеров диалогового окна `UserForm`
`Scroll`	Прокрутка диалогового окна `UserForm`
`Terminate`	Закрытие диалогового окна `UserForm`
`Zoom`	Изменение масштаба диалогового окна `UserForm`

Перекрестная ссылка

Во многих примерах из глав 13–15 демонстрируется обработка событий для пользовательских диалоговых окон и находящихся в них элементов управления.

События, не связанные с объектами

События, которые рассматривались ранее, были связаны с конкретными объектами (`Application`, `Workbook`, `Sheet` и т.д.). В этом разделе описаны дополнительные события, которые не связаны с определенными объектами: `OnTime` и `OnKey`. Доступ к ним осуществляется с помощью методов объекта `Application`.

Примечание

В отличие от других событий, которые рассматриваются в этой главе, события `OnTime` и `OnKey` программируются с помощью кода, расположенного в модуле VBA общего назначения.

Событие `OnTime`

Событие `OnTime` происходит в определенное время суток. В следующем примере демонстрируется обработка событий Excel таким образом, что ровно в 15:00 отображается окно сообщения и раздается звуковой сигнал.

```
Sub SetAlarm()
  Application.OnTime TimeValue("15:00:00"), "DisplayAlarm"
End Sub

Sub DisplayAlarm()
  Beep
  MsgBox "Просыпайся. Пришло время пообедать!"
End Sub
```

В представленном примере процедура `SetAlarm` использует метод `OnTime` объекта `Application` для настройки события `OnTime`. Этот метод принимает два аргумента: время (в данном случае — 3:00 p.m., или 15:00) и имя процедуры, которая будет запущена в указанное время (в приведенном примере используется процедура `DisplayAlarm`). После выполнения процедуры `SetAlarm` процедура `DisplayAlarm` будет вызвана в 15:00, что приведет к отображению окна сообщения, показанного на рис. 19.10.

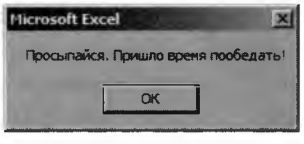


Рис. 19.10. Это сообщение появляется в заранее выбранное время

Если необходимо запланировать событие относительно текущего времени суток (например, через 20 минут), то воспользуйтесь следующим оператором.

```
Application.OnTime Now + TimeValue("00:20:00"), _
"DisplayAlarm"
```

Метод `OnTime` можно также использовать, если вы планируете вызов процедуры на определенную дату. Представленный далее оператор запустит процедуру `DisplayAlarm` 1 апреля 2010 года в 12:01.

```
Application.OnTime DateSerial(2008, 4, 1) + _
   TimeValue("00:00:01"), "DisplayAlarm"
```

Примечание

Метод `OnTime` включает два дополнительных аргумента. Если вы планируете использовать этот метод, обратитесь к справочному руководству для получения более полной информации.

Приведенные ниже две процедуры демонстрируют методы управления повторяющимися событиями. В данном случае ячейка A1 обновляется каждые пять секунд и в нее заносится текущее время. При выполнении процедур `UpdateClock` время сохраняется в ячейке A1, и эта операция повторяется через пять секунд. Для остановки "часов" используется процедура `StopClock`. Обратите внимание, что `NextTick` — это переменная уровня модуля, которая сохраняет интервал времени перед наступлением следующего события.

Компакт-диск

Рассматриваемый в этом разделе пример находится на прилагаемом к книге компакт-диске в файле `ontime event demo.xlsm`.

```
Dim NextTick As Date
Sub UpdateClock()
'   Вносит в ячейку A1 текущее значение времени
   ThisWorkbook.Sheets(1).Range("A1") = Time
'   Следующее событие вызывается через пять секунд
   NextTick = Now + TimeValue("00:00:05")
   Application.OnTime NextTick, "UpdateClock"
End Sub

Sub StopClock()
'   Отменяет событие OnTime (часы останавливаются)
   On Error Resume Next
   Application.OnTime NextTick, "UpdateClock", , False
End Sub
```

Предупреждение

Событие `OnTime` наступает даже в том случае, когда рабочая книга закрывается. Другими словами, если закрыть рабочую книгу без запуска процедуры `StopClock`, рабочая книга повторно откроется через пять секунд (если Excel будет выполняться). Для предотвращения подобного поведения рабочей книги используется процедура обработки событий `Workbook_BeforeClose`, которая содержит следующий оператор:

```
Call StopClock
```

Перекрестная ссылка

Для просмотра примера повторения события `OnTime` обратитесь к примеру стрелочных часов, рассмотренному в главе 18.

Событие OnKey

В процессе работы Excel постоянно отслеживает нажимаемые пользователем клавиши. Поэтому вы можете настроить клавишу (или комбинацию клавиш), которая после нажатия запустит определенную процедуру. Эта клавиша не распознается, если пользователь вводит формулу или работает в диалоговом окне.

Предупреждение

Учтите, что процедура, которая вызывается в ответ на событие OnKey, не ограничивается единственной рабочей книгой. Выбранная пользователем комбинация клавиш может применяться во всех открытых рабочих книгах. Также, если вы планируете работать с событием OnKey, убедитесь в том, что имеется способ отмены вызываемого события. Для этого обычно используется процедура обработки события Workbook_BeforeClose.

Пример использования события OnKey

В следующем примере используется метод OnKey для настройки события OnKey. В данном случае переназначается значение клавиш <PgUp> и <PgDn>. После выполнения процедуры Setup_OnKey нажатие клавиши <PgDn> приведет к запуску процедуры PgDn_Sub, а нажатие клавиши <PgUp> — процедуры PgUp_Sub. В итоге проведенных изменений нажатие клавиши <PgDn> вызывает перемещение курсора на одну строку вниз, а клавиши <PgUp> — на одну строку вверх.

```
Sub Setup_OnKey()
   Application.OnKey "{PgDn}", "PgDn_Sub"
   Application.OnKey "{PgUp}", "PgUp_Sub"
End Sub

Sub PgDn_Sub()
   On Error Resume Next
   ActiveCell.Offset(1, 0).Activate
End Sub

Sub PgUp_Sub()
   On Error Resume Next
   ActiveCell.Offset(-1, 0).Activate
End Sub
```

Компакт-диск

Рассматриваемый в этом разделе пример находится на прилагаемом к книге компакт-диске в файле onkey event demo.xlsm.

В предыдущих примерах использовался оператор On Error Resume Next, который позволял игнорировать все возникающие ошибки. Например, если активная ячейка находится в первой строке, то перемещение на одну строку вверх приводит к возникновению ошибки. Кроме того, если активным является лист диаграммы, то возникнет ошибка, так как на листе диаграммы не существует такого понятия, как активная ячейка.

Запустив следующую процедуру, можно отменить событие OnKey и вернуть клавишам их обычную функциональность.

```
Sub Cancel_OnKey()
   Application.OnKey "{PgDn}"
```

```
   Application.OnKey "{PgUp}"
End Sub
```

Использование пустой строки в качестве второго аргумента метода `OnKey` не приводит к отмене текущего события `OnKey`. Вместо этого Excel игнорирует нажатие указанных клавиш и не выполняет вообще никаких действий при их нажатии. Например, приведенный ниже оператор сообщает Excel, что необходимо игнорировать нажатие комбинации клавиш <Alt+F4> (символ процента представляет клавишу <Alt>).

```
Application.OnKey "%{F4}", ""
```

Перекрестная ссылка

Хотя метод `OnKey` можно использовать для назначения комбинаций клавиш макросу, лучше для этого обратиться к диалоговому окну Параметры макроса (Macro Options). Дополнительные сведения можно найти в главе 9.

Коды клавиш

Наверное, вы обратили внимание, что в предыдущем примере кода клавиша <PgDn> заключена в фигурные скобки. Это — код клавиши. В табл. 19.6 приведены коды клавиш, которые можно использовать в процедурах, обрабатывающих событие `OnKey`.

Таблица 19.6. Коды клавиш для события `OnKey`

Клавиша	Код
<Backspace>	{BACKSPACE} или {BS}
<Break>	{BREAK}
<Caps Lock>	{CAPSLOCK}
<Delete> или <Del>	{DELETE} или {DEL}
<↓>	{DOWN}
<End>	{END}
<Enter>	~ (тильда)
<Enter> (на цифровой клавиатуре)	{ENTER}
<Escape>	{ESCAPE} или {ESC}
<Home>	{HOME}
<Ins>	{INSERT}
<←>	{LEFT}
<NumLock>	{NUMLOCK}
<Page Down>	{PGDN}
<Page Up>	{PGUP}
<→>	{RIGHT}
<Scroll Lock>	{SCROLLLOCK}
<Tab>	{TAB}
<↑>	{UP}
От <F1> до <F15>	От {F1} до {F15}

Можно также определить комбинации клавиш совместно с клавишами <Shift>, <Ctrl> и <Alt>. Для определения комбинации клавиш совместно с другими клавишами воспользуйтесь следующим синтаксисом:

- **<Shift>** — знак "плюс" (+);
- **<Ctrl>** — знак "крышка" (^);
- **<Alt>** — знак процента (%).

Например, для назначения процедуре комбинации клавиш <Ctrl+Shift+A> воспользуйтесь следующим оператором:

```
Application.OnKey "^+A", "SubName:"
```

Для назначения процедуре комбинации клавиш <Alt+F11> (обычно эти клавиши применяются для открытия окна VB Editor) используйте следующий оператор:

```
Application.OnKey "^{F11}", "SubName"
```

Отключение контекстного меню

Ранее уже рассматривалась процедура `Worksheet_BeforeRightClick`, которая отключает контекстное меню. Приведенная ниже процедура размещается в модуле кода объекта `ЭтаКнига` (`ThisWorkbook`).

```
Private Sub Worksheet_BeforeRightClick _
   (ByVal Target As Excel.Range, Cancel As Boolean)
   Cancel = True
   MsgBox "Попытка - не пытка, но вам не повезло."
End Sub
```

Пользователь по-прежнему может вызывать контекстное меню с помощью комбинации клавиш <Shift+F10>. Для перехвата кодов этих клавиш в стандартный модуль VBA добавьте следующие процедуры.

```
Sub SetupNoShiftF10()
   Application.OnKey "+{F10}", "NoShiftF10"
End Sub

Sub TurnOffNoShiftF10()
   Application.OnKey "+{F10}"
End Sub

Sub NoShiftF10()
   MsgBox "Эти клавиши не работают."
End Sub
```

После вызова на выполнение процедуры `SetupNoShiftF10` нажмите комбинацию клавиш <Shift+F10> — и вы увидите сообщение, показанное на рис. 19.11. Помните о том, что процедура `Worksheet_BeforeRightClick` корректно работает только в той рабочей книге, в которой она была создана. С другой стороны, событие, вызываемое нажатием комбинации клавиш <Shift+F10>, применяется ко всем открытым рабочим книгам.

Примечание

На некоторых клавиатурах имеется специальная клавиша, которая вызывает контекстное меню. На моей клавиатуре такая клавиша находится в правой части клавиатуры между клавишами <Windows> и <Ctrl>. Как ни странно, но процедура перехвата комбинации клавиш <Shit+F10> также отключает выделенную клавишу вызова контекстного меню.

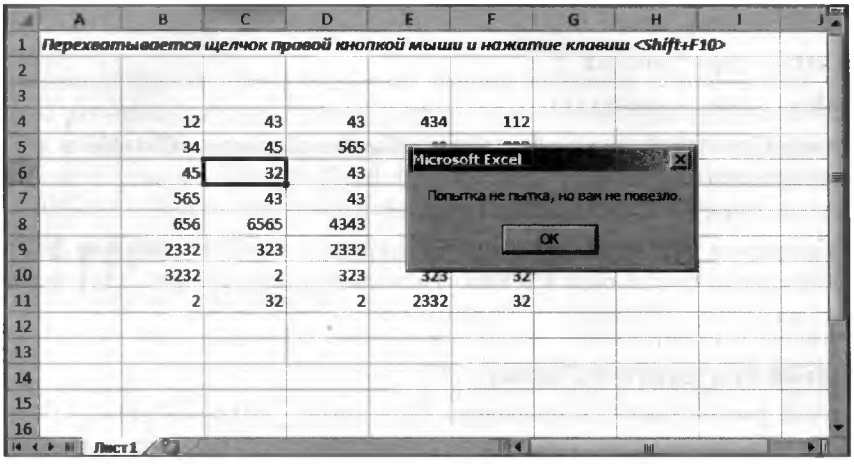


Рис. 19.11. Это сообщение отображается после нажатия комбинации клавиш <Shift+F10>

Компакт-диск

На прилагаемом к книге компакт-диске находится рабочая книга, которая включает все описанные в этом разделе процедуры. Файл под названием `no shortcut menus.xlsm` включает процедуры обработки событий: процедура `Workbook_Open` вызывает процедуру `SetupNoShiftF10`, а процедура `Workbook_BeforeClose` вызывает процедуру `TurnOffNoShiftF10`.

Глава 20

Взаимодействие с другими приложениями

В этой главе...

- Запуск другого приложения из Excel
- Активизация другого приложения с помощью Excel
- Запуск аплетов папки Панель управления и мастеров
- Автоматизация
- Отправка почтовых сообщений с помощью Outlook
- Отправка почтовых вложений с помощью Excel
- Использование метода `SendKeys`

В данной главе рассмотрены способы взаимодействия приложений Excel с другими приложениями. Кроме того, вы ознакомитесь с полезными примерами.

Запуск другого приложения из Excel

Иногда необходимо запустить другое приложение из Excel, например вызвать на выполнение другую программу, запустить программу установки соединения или даже командный файл DOS. Разработчик приложения также может облегчить пользователю доступ к папке Панель управления.

Использование функции `Shell`

Функция `Shell` упрощает запуск других приложений. Приведенная ниже процедура `StartCalc` используется для открытия калькулятора Windows.

```
Sub StartCalc()
  Dim Program As String
```

```
    Dim TaskID As Double
    On Error Resume Next
    Program = "calc.exe"
    TaskID = Shell(Program, 1)
    If Err <> 0 Then
       MsgBox "Невозможно запустить " & Program, vbCritical, "Ошибка"
    End If
End Sub
```

Результат выполнения этой процедуры представлен на рис. 20.1.

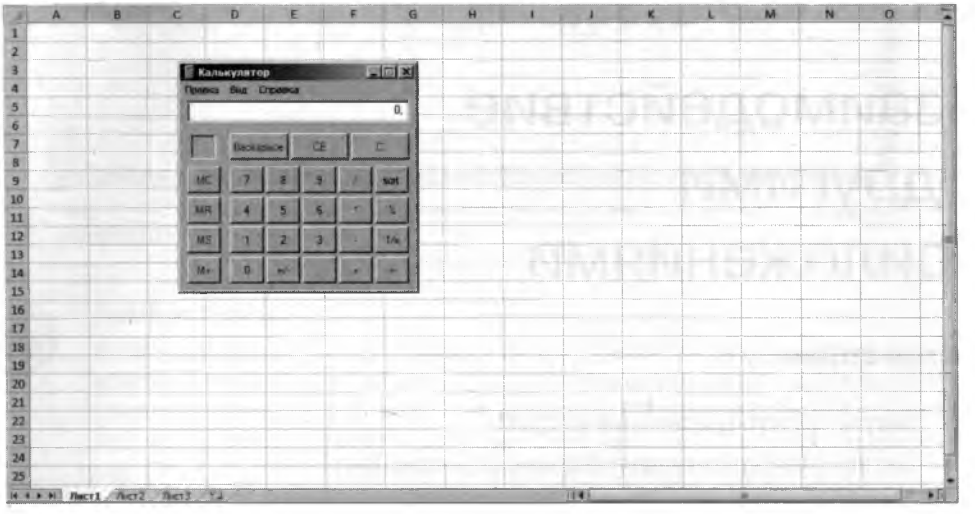


Рис. 20.1. Запуск калькулятора Windows из Excel

Функция `Shell` возвращает номер идентификатора приложения. Этот номер можно использовать позднее для активизации задачи. Второй аргумент функции `Shell` определяет способ отображения приложения (для окна нормального размера с последующей активизацией используется аргумент 1). Описание других значений этого аргумента можно найти в справочной системе.

Если вызов функции `Shell` завершился неудачно, то будет создано сообщение об ошибке. Таким образом, эта процедура использует оператор `On Error` для отображения сообщения, если файл нельзя найти или произошла другая ошибка.

Отображение окна папки

Функция `Shell` удобна, когда нужно отобразить выбранную папку в окне проводника Windows. Например, следующий оператор отображает на экране папку, в которой находится активная рабочая книга (если книга была предварительно сохранена в этой папке).

```
If ActiveWorkbook.Path <> "" Then _
   Shell "explorer.exe " & ActiveWorkbook.Path, vbNormalFocus
```

Важно понимать, что код VBA продолжает свою работу после запуска приложения, загруженного с помощью функции `Shell`. Другими словами, функция `Shell` запускает приложения *асинхронно*. Если в процедуре после вызова функции `Shell` находятся другие инструкции, то они выполняются одновременно с запущенной программой. Если ин-

струкции требуют вмешательства пользователя (например, при отображении окна сообщения), то строка заголовка Excel будет мигать, пока активно другое приложение.

В некоторых случаях может понадобиться запустить приложение с помощью функции `Shell`, но при этом выполнение кода VBA должно прекратиться, пока запущенное приложение не создаст файл, который будет использован далее в коде VBA. Несмотря на то что выполнение кода приостановить невозможно, вы вправе создать цикл, который, кроме отслеживания состояния запущенного приложения, не выполняет никаких функций. Ниже приведен пример окна сообщения, которое отображается на экране тогда, когда приложение, запущенное с помощью функции `Shell`, завершает свою работу.

```
Declare Function OpenProcess Lib "kernel32" _
   (ByVal dwDesiredAccess As Long, _
   ByVal bInheritHandle As Long, _
   ByVal dwProcessId As Long) As Long
Declare Function GetExitCodeProcess Lib "kernel32" _
   (ByVal hProcess As Long, _
   lpExitCode As Long) As Long
Sub StartCalc2()
   Dim TaskID As Long
   Dim hProc As Long
   Dim lExitCode As Long
   Dim ACCESS_TYPE As Integer, STILL_ACTIVE As Integer
   Dim Program As String
   ACCESS_TYPE = &H400
   STILL_ACTIVE = &H103
   Program = "Calc.exe"
   On Error Resume Next
'  Определение оболочки
   TaskID = Shell(Program, 1)
'  Получение дескриптора процесса
   hProc = OpenProcess(ACCESS_TYPE, False, TaskID)

   If Err <> 0 Then
      MsgBox "Невозможно запустить " & Program, vbCritical, "Ошибка"
      Exit Sub
   End If

   Do ' Непрерывный цикл
'     Проверка процесса
      GetExitCodeProcess hProc, lExitCode
'     Разрешить обработку события
      DoEvents
   Loop While lExitCode = STILL_ACTIVE

'  Задача завершена, отображение сообщения

   MsgBox Program & " закрыто"
End Sub
```

После запуска приложения данная процедура будет постоянно вызывать функцию `GetExitCodeProcess` в конструкции `Do Loop`. В цикле проверяется значение переменной `lExitCode`. Когда программа завершает работу, `lExitCode` изменяет свое значение, цикл завершается, и код VBA продолжает выполняться.

Компакт-диск

Оба рассматриваемых в разделе примера можно найти на прилагаемом к книге компакт-диске в файле `start calculator.xlsm`.

Использование API-функции `ShellExecute`

Функция Windows API `ShellExecute` обеспечивает запуск других приложений. Примечательно, что другое приложение запускается только в том случае, когда открываемый файл имеет один из зарегистрированных в системе типов. Например, можете воспользоваться функцией `ShellExecute` для открытия веб-документа в окне браузера, выбранного по умолчанию. Либо, щелкнув на электронном адресе, можете открыть почтовый клиент, заданный по умолчанию.

Ниже представлено объявление этой API-функции (этот код работает только в Excel 2010).

```
Private Declare PtrSafe 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
```

Следующая процедура демонстрирует вызов функции `ShellExecute`. В данном случае с ее помощью запускается графический редактор, посредством которого в системе просматриваются файлы в формате JPG. Если возвращаемый функцией результат меньше 32, отображается сообщение об ошибке.

```
Sub ShowGraphic()
   Dim FileName As String
   Dim Result As Long
   FileName = ThisWorkbook.Path & "\flower.jpg"
   Result = ShellExecute(0&, vbNullString, FileName, _
      vbNullString, vbNullString, vbNormalFocus)
   If Result < 32 Then MsgBox "Ошибка"
End Sub
```

Приведенная ниже процедура открывает текстовый файл с помощью заданного по умолчанию текстового редактора.

```
Sub OpenTextFile()
   Dim FileName As String
   Dim Result As Long
   FileName = ThisWorkbook.Path & "\textfile.txt"
   Result = ShellExecute(0&, vbNullString, FileName, _
      vbNullString, vbNullString, vbNormalFocus)
   If Result < 32 Then MsgBox "Ошибка"
End Sub
```

Ниже приводится похожий пример, но здесь открывается URL-ссылка с помощью заданного по умолчанию браузера.

```
Sub OpenURL()
   Dim URL As String
   Dim Result As Long
   URL = "http://spreadsheetpage.com"
   Result = ShellExecute(0&, vbNullString, URL, _
      vbNullString, vbNullString, vbNormalFocus)
   If Result < 32 Then MsgBox "Ошибка"
End Sub
```

Эта же методика применяется и по отношению к почтовым сообщениям. В приведенном ниже примере создается сообщение по умолчанию, в котором указан адрес получателя. Это сообщение рассылается получателям с помощью заданного по умолчанию почтового клиента.

```
Sub StartEmail()
   Dim Addr As String
   Dim Result As Long
   Addr = "mailto:bgates@microsoft.com"
   Result = ShellExecute(0&, vbNullString, Addr, _
      vbNullString, vbNullString, vbNormalFocus)
   If Result < 32 Then MsgBox "Ошибка"
End Sub
```

Компакт-диск

Описанные в разделе примеры доступны на прилагаемом к книге компакт-диске в файле `shellexecute examples.xlsm`.

Активизация другого приложения с помощью Excel

В предыдущих разделах рассматривались различные способы запуска приложения на выполнение. При этом возможна следующая проблема: может оказаться, что приложение уже выполняется в момент использования функции `Shell` для его запуска. Это приведет к запуску нового сеанса указанного приложения. Вам же требуется перейти к уже запущенному сеансу работы программы, а не создать новый.

Инструкция `AppActivate`

В представленной ниже процедуре `StartCalculator` инструкция `AppActivate` применяется для активизации выполняющегося приложения (в данном случае идет речь о приложении калькулятора). В качестве аргумента используется заголовок приложения. Если инструкция `AppActivate` возвращает ошибку, то приложение калькулятора еще не запущено. Далее процедура запускает указанное приложение.

```
Sub StartCalculator()
   Dim AppFile As String
   Dim CalcTaskID As Double

   AppFile = "Calc.exe"
   On Error Resume Next
   AppActivate "Calculator"
   If Err <> 0 Then
      Err = 0
      CalcTaskID = Shell(AppFile, 1)
      If Err <> 0 Then MsgBox "Невозможно открыть калькулятор"
   End If
End Sub
```

Компакт-диск

Рассматриваемый в этом разделе пример находится на прилагаемом к книге компакт-диске в файле `start calculator.xlsm`.

Активизация приложения Microsoft Office

Если запускаемое приложение входит в состав пакета Microsoft Office, то вы можете использовать метод `ActivateMicrosoftApp` объекта `Application`. Например, следующая процедура запускает программу Word.

```
Sub StartWord()
   Application.ActivateMicrosoftApp xlMicrosoftWord
End Sub
```

Если программа Word уже запущена, то предыдущая процедура просто активизирует его. В используемом методе можно применять и другие константы:

- `xlMicrosoftPowerPoint`;
- `xlMicrosoftMail (activates Outlook)`;
- `xlMicrosoftAccess`;
- `xlMicrosoftFoxPro`;
- `xlMicrosoftProject`;
- `xlMicrosoftSchedulePlus`.

Запуск аплетов папки Панель управления и мастеров

Windows предоставляет в распоряжение пользователя большое количество системных диалоговых окон и мастеров, многие из которых расположены в окне Панель управления (Control Panel). В приложении Excel может понадобиться отобразить одно или несколько диалоговых окон из этой папки. Например, вы имеете возможность отобразить диалоговое окно Свойства: Дата и время (Windows Date and Time Properties), которое показано на рис. 20.2.

Рис. 20.2. С помощью VBA можно отобразить окно аплета панели управления

Для того чтобы открыть другие системные диалоговые окна, следует запустить приложение `rundll32.exe` с помощью функции `Shell`.

Следующая процедура открывает диалоговое окно **Свойства: Дата и время** (Windows Date and Time Properties).

```
Sub ShowDateTimeDlg()
   Dim Arg As String
   Dim TaskID As Double
   Arg = "rundll32.exe shell32.dll,Control_RunDLL timedate.cpl"
   On Error Resume Next
   TaskID = Shell(Arg)
   If Err <> 0 Then
      MsgBox ("Невозможно запустить приложение.")
   End If
End Sub
```

Ниже приведен общий формат вызова приложения `rundll32.exe`.

```
rundll32.exe shell32.dll,Control_RunDLL имя_файла.cpl, n, t
```

- *имя_файла.cpl* — имя одного из файлов `*.CPL` на панели управления;
- *n* — номер аплета в файле `*.CPL` (начинается с нуля);
- *t* — количество вкладок (для диалоговых окон, включающих несколько вкладок).

Компакт-диск

Рабочая книга, демонстрирующая 12 дополнительных аплетов панели управления, показанных на рис. 20.3, находится на прилагаемом компакт-диске в файле `control panel dialogs.xlsm`.

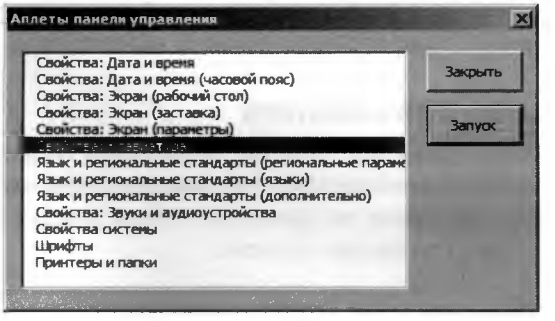


Рис. 20.3. Рабочая книга, отображающая данное диалоговое окно, демонстрирует методы открытия системных диалоговых окон

Автоматизация

Можно создать макрос Excel, который будет управлять программой Microsoft Word. Точнее, макрос Excel будет контролировать самый важный компонент Word — так называемый сервер автоматизации. В подобных случаях Excel выступает *клиентским*, а Word — *серверным приложением*. Более того, можно создать приложение Visual Basic, которое управляет работой Excel. Процесс управления одного приложения другим часто называют *OLE-автоматизацией*, или простой автоматизацией.

Под автоматизацией подразумевается достаточно привлекательная технология. Разработчик, которому необходимо создать диаграмму, может обратиться к другому приложению, получить от него объект `Chart` и использовать свойства и методы этого объек-

та. Автоматизация в некотором смысле размывает границы между приложениями. Конечный пользователь может работать с объектом Access и даже не подозревать об этом.

Примечание

Одни приложения, в том числе Excel, могут быть как серверными, так и клиентскими, а другие могут быть клиентскими либо серверными.

В этом разделе продемонстрированы способы использования VBA для получения доступа к объектам, которые предоставляются другими приложениями, и для управления объектами. В приведенных примерах используется только программа Microsoft Word, но рассматриваемая концепция в одинаковой степени относится ко всем приложениям, которые предоставляют собственные объекты для автоматизации (с течением времени таких приложений становится все больше).

Работа с внешними объектами

В Excel можно использовать команду Вставка⇨Текст⇨Объект (Insert⇨Text⇨Object) для встраивания объекта (например, документа Word) в рабочий лист. Кроме того, существует возможность создать объект и манипулировать им в VBA. (Это возможно благодаря все той же технологии автоматизации.) При таком подходе программе предоставляется полный доступ к объекту. Разработчикам подобная технология обычно нравится больше, чем встраивание объекта в рабочий лист. Когда объект встроен, пользователь должен точно знать, как пользоваться приложением, которому принадлежит объект. Но если для управления объектом применяется код VBA, можно настроить объект так, что пользователь будет управлять им с помощью таких простых операций, как щелчок на кнопке.

Раннее и позднее связывание

Перед началом работы с внешним объектом необходимо создать его экземпляр. Это можно сделать с помощью раннего или позднего связывания. *Связывание* обозначает установку соответствия между вызовами тех функций, которые создает разработчик, и фактическим кодом, выполняемым при запуске функции.

Раннее связывание

Для использования раннего связывания создайте ссылку на объектную библиотеку с помощью команды VBE Tools⇨References (Сервис⇨Ссылки). Откроется диалоговое окно, показанное на рис. 20.4. Установите флажок, соответствующий объектной библиотеке, на которую установлена ссылка.

После создания ссылки на библиотеку объектов можно использовать окно Object Browser (показанное на рис. 20.5) для просмотра имен, методов и свойств объектов. Для открытия этого окна нажмите клавишу <F2> в среде VBE.

При использовании раннего связывания следует создать ссылку на конкретную версию библиотеки объектов. Например, можно указать Microsoft Word 10.0 Object Library (для Word 2002), Microsoft Word 11.0 Object Library (для Word 2003) или Microsoft Word 14.0 Object Library (для Word 2010). После этого используется следующий оператор для создания объекта:

```
Dim WordApp As New Word.Application
```

Рис. 20.4. Добавление ссылки на файл объектной библиотеки

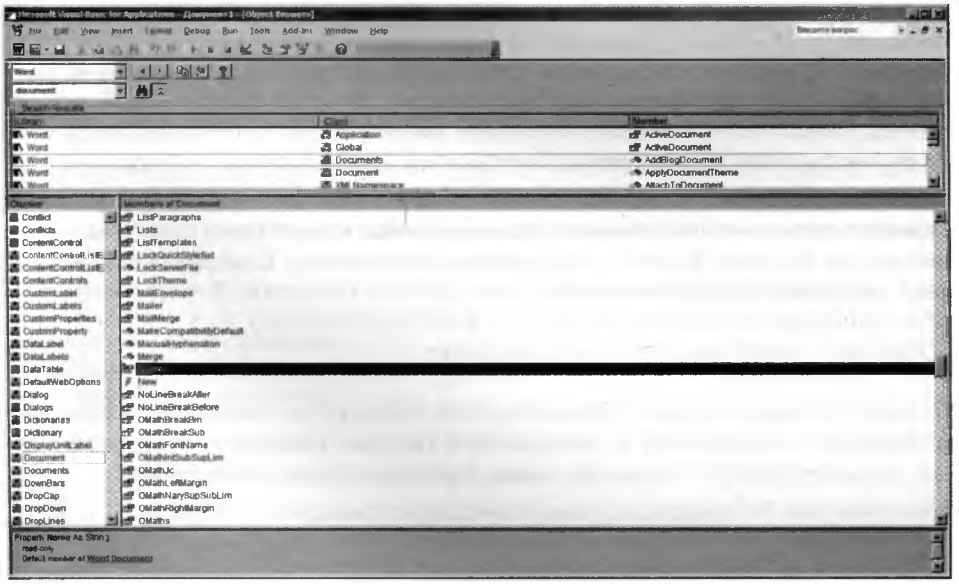


Рис. 20.5. Используйте окно Object Browser для получения сведений об объектах, находящихся в библиотеке

Применение раннего связывания для создания объекта с помощью ссылки на библиотеку объектов является более эффективным методом, который обеспечивает большую производительность приложения. Раннее связывание может использоваться только тогда, когда контролируемый объект имеет отдельный файл библиотеки типов или библиотеки объектов. Кроме того, следует удостовериться, что в системе пользователя установлена необходимая версия библиотеки объектов.

Другим преимуществом раннего связывания является то, что оно позволяет применять константы, которые объявлены в библиотеке объектов. Например, Word (как и Excel) содержит ряд предопределенных констант, которые можно использовать в коде VBA. При использовании раннего связывания можно вводить эти константы в создавае-

мом коде. Если используется позднее связывание, вам придется обратиться к фактическому значению константы, а не к ее имени.

Еще одним преимуществом раннего связывания является возможность использования окна Object Browser и параметра Auto List Members (Автоматическая вставка объектов) в редакторе VBE для получения простого доступа к свойствам и методам объектов. Подобная возможность недоступна при использовании позднего связывания, поскольку тип объекта определяется только на этапе выполнения.

Позднее связывание

На этапе выполнения можно использовать или функцию `CreateObject` для создания объекта, или функцию `GetObject` для получения сохраненного объекта. Такие объекты объявляются как объекты универсального типа `Object`. Ссылка на объекты задается на этапе выполнения.

Функции `GetObject` и `CreateObject`

Обе функции VBA, `GetObject` и `CreateObject`, возвращают ссылки на объект, хотя работают совершенно по-разному.

Функция `CreateObject` применяется для создания интерфейса нового экземпляра приложения. Ее следует использовать в том случае, когда приложение еще не запущено. Если экземпляр приложения уже выполняется, то будет создан новый экземпляр. Например, представленный далее оператор загружает Excel и возвращает для переменной `XLApp` ссылку на созданный объект `Excel.Application`.

```
Set XLApp = CreateObject("Excel.Application")
```

Функция `GetObject` используется приложением, которое уже запущено, или применяется для загрузки файла в запускаемом приложении. Следующий оператор, например, загружает Excel и загружает в него файл `Myfile.xls`. В переменную `XLBook` возвращается ссылка на объект `Workbook` (файл `Myfile.xlsx`).

```
Set XLBook = GetObject("C:\Myfile.xlsx")
```

Позднее связывание можно применить даже тогда, когда неизвестна версия библиотеки объектов, установленной в операционной системе. Например, следующий код, который управляет Word 97 и более поздними версиями, создает объект Word.

```
Dim WordApp As Object
Set WordApp = CreateObject("Word.Application")
```

Если в системе установлено несколько версий Word, то можно создать объект определенной версии. Приведенный ниже оператор управляет объектом Word 2003.

```
Set WordApp = CreateObject("Word.Application.11")
```

Параметр в реестре Windows для объекта `Automation` программы Word, а также ссылка на объект `Application` в VBA оказались одинаковыми: `Word.Application`. Но они указывают на разные элементы. Когда объект объявляется с помощью оператора `As Word.Application` или `As New Word.Application`, то этот термин обозначает объект `Application` в библиотеке Word. А если вызвать функцию `CreateObject("Word.Application")`, то термин будет ссылаться на параметр, под которым последняя версия Word известна в системном реестре Windows. Данное утверждение не относится ко всем объектам автоматизации, хотя и справедливо для основных компонентов пакета Office 2010. Если пользователь заменит Word 2003 на Word 2010, то функция `CreateObject("Word.Application")` будет действительной — она ссы-

лается на новое приложение. Но если удалить Word 2010, то функция `CreateObject("Word.Application.14")`, обращающаяся непосредственно к Word 2010, работать не будет.

Функция `CreateObject`, которая обращается к таким объектам автоматизации, как `Word.Application` или `Excel.Application`, всегда возвращает *новый экземпляр* объекта, т.е. запускается новая копия приложения. Поэтому в случае, когда объект автоматизации уже запущен, запускается новый экземпляр, после чего создается новый объект указанного типа.

Для того чтобы использовать текущий экземпляр или чтобы запустить приложение и заставить его загрузить сохраненный файл, воспользуйтесь функцией `GetObject`.

Примечание

Для использования автоматизации в приложениях Office рекомендуется выполнить раннее связывание с самой ранней версией программного продукта, который установлен в системе. Например, чтобы реализовать автоматизацию с Word 2003, Word 2007 и Word 2010, следует использовать библиотеку объектов Word 2003 для обеспечения совместимости со всеми тремя версиями. Таким образом, гарантируется невозможность использования уникальных функций, предоставляемых более поздними версиями Word.

Простой пример позднего связывания

Приведенный пример демонстрирует создание объекта Word с помощью позднего связывания. Рассматриваемая процедура создает объект, отображает номер версии, закрывает приложение Word и уничтожает созданный объект (освобождая используемую память).

```
Sub GetWordVersion()
   Dim WordApp As Object
   Set WordApp = CreateObject("Word.Application")
   MsgBox WordApp.Version
   WordApp.Quit
   Set WordApp = Nothing
End Sub
```

Примечание

Созданный с помощью приведенной выше процедуры объект Word является невидимым. Если нужно, чтобы в процессе обработки этот объект отображался на экране, присвойте свойству `Visible` значение `True`, как показано ниже.

```
WordApp.Visible = True
```

Данный пример можно запрограммировать с помощью раннего связывания. Однако вначале воспользуйтесь командой Tools⇨References (Сервис⇨Ссылки) для создания ссылки на библиотеку объектов Word. После этого можете применить приведенный ниже код.

```
Sub GetWordVersion()
   Dim WordApp As New Word.Application
   MsgBox WordApp.Version
   WordApp.Quit
   Set WordApp = Nothing
End Sub
```

Управление приложением Word из Excel

В примере, представленном в этом разделе, отображается сеанс автоматизации программы Word в Excel. Процедура MakeMemos создает три заметки в Word, после чего сохраняет их в отдельных файлах. Информация, которая использовалась для создания заметок, показана на рис. 20.6.

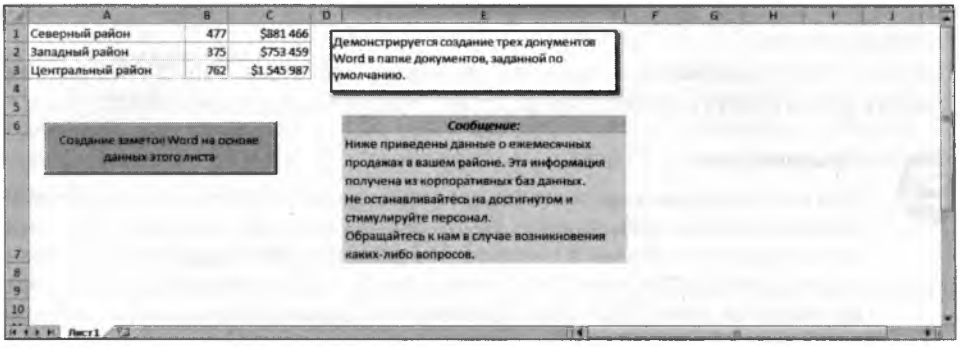


Рис. 20.6. Программа Word автоматически генерирует три заметки на основе данных Excel

Процедура MakeMemos начинается с запуска объекта WordApp. После этого просматриваются три строки данных на листе Лист1 и используются свойства и методы объекта Word для создания каждой заметки и ее сохранения в отдельном файле. Диапазон Message (в ячейке E6) содержит текст, который применяется для создания заметок. Операции выполняются в фоновом режиме и окно Word на экране не отображается.

```
Sub MakeMemos()
'   Создание заметок в Word с помощью автоматизации
    Dim WordApp As Object
    Dim Data As Range, message As String
    Dim Records As Integer, i As Integer
    Dim Region As String, SalesAmt As String, SalesNum As String
    Dim SaveAsName As String
'   Запуск Word и создание объекта (позднее связывание)
    Set WordApp = CreateObject("Word.Application")
'   Информация с рабочего листа
    Set Data = Sheets("Лист1").Range("A1")
    Message = Sheets("Лист1").Range("Message")

'   Циклический просмотр всех записей на листе Лист1
    Records = Application.CountA(Sheets("Лист1").Range("A:A"))
    For i = 1 To Records
'       Обновление сообщения в строке состояния
        Application.StatusBar = "Обработка записи " & i
'       Назначение данных переменным
        Region = Data.Cells(i, 1).Value
        SalesNum = Data.Cells(i, 2).Value
        SalesAmt = Format(Data.Cells(i, 3).Value, "#,000")

'       Определение имени файла
        SaveAsName = Application.DefaultFilePath & _
            "\" & Region & ".docx"
'       Передача команд в Word
        With WordApp
```

```
        .Documents.Add
        With .Selection
            .Font.Size = 14
            .Font.Bold = True
            .ParagraphFormat.Alignment = 1
            .TypeText Text:="М Е М О Р А Н Д У М"
            .TypeParagraph
            .TypeParagraph
            .Font.Size = 12
            .ParagraphFormat.Alignment = 0
            .Font.Bold = False
            .TypeText Text:="Дата:" & vbTab & _
               Format(Date, "d mmmm, yyyy")
            .TypeParagraph
            .TypeText Text:="Получатель:" & vbTab & _
               "Менеджер, " & Region
            .TypeParagraph
            .TypeText Text:="Отправитель:" & vbTab & _
               Application.UserName
            .TypeParagraph
            .TypeParagraph
            .TypeText Message
            .TypeParagraph
            .TypeParagraph
            .TypeText Text:="Количество проданных позиций:" _
               & vbTab & SalesNum
            .TypeParagraph
            .TypeText Text:="Сумма:" & vbTab & _
               Format(SalesAmt, "$#,##0")
        End With
        .ActiveDocument.SaveAs FileName:=SaveAsName
    End With
  Next i
' Удаление объекта
  WordApp.Quit
  Set WordApp = Nothing
' Обновление строки состояния
  Application.StatusBar = ""
  MsgBox Records & " заметок создано и сохранено в " & _
     Application.DefaultFilePath
End Sub
```

На рис. 20.7 показан один из документов, созданных с помощью процедуры MakeMemos.

Компакт-диск

Рассматриваемая в этом разделе рабочая книга находится на прилагаемом компакт-диске в файле make memos.xlsm.

Этот макрос создавался в несколько этапов. Все началось с его записи в Word. Записывались действия по созданию нового документа, добавлению и форматированию текста, а также сохранению файла. Этот макрос Word предоставил необходимую информацию о свойствах и методах объекта Word. После этого макрос был скопирован в модуль кода Excel. Обратите внимание, что использовался оператор With-End With. Перед каждой инструкцией между With и End With была добавлена точка. Например, первоначальный макрос Word содержал (среди прочих) приведенный ниже оператор.

```
Documents.Add
```

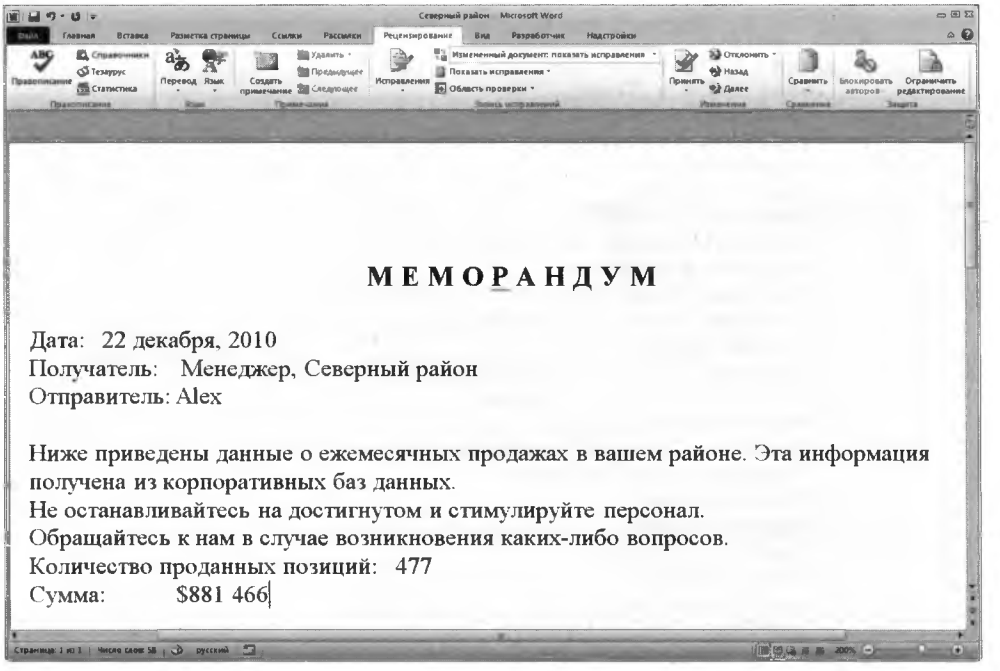


Рис. 20.7. Этот документ Word создан с помощью макроса Excel

Затем макрос был изменен следующим образом.

```
With WordApp
    .Documents.Add
'    Здесь находятся дополнительные инструкции
End With
```

Макрос, записанный в Word, использовал несколько встроенных констант. В связи с тем, что в данном примере используется позднее связывание, пришлось заменить константы их фактическими значениями. Значения можно узнать в окне отладки (Immediate) редактора VBE в Word.

Управление программой Excel из другого приложения

Вы имеете возможность управлять Excel из другого приложения (из программы Visual Basic или из макроса Word); например, если необходимо провести некоторые расчеты в Excel, а результаты передать обратно в документ Word.

Создайте один из следующих объектов Excel с помощью указанной рядом функции:

- объект `Application` — функция `CreateObject("Excel.Application");`
- объект `Workbook` — функция `CreateObject("Excel.Sheet");`
- объект `Chart` — функция `CreateObject("Excel.Chart").`

Ниже показан код процедуры, которая находится в модуле VBA документа Word 2010. Эта процедура создает объект Excel `Worksheet` (представленный параметром `"Excel.Sheet"`) на основе существующей рабочей книги и вставляет его в файл Word.

```
Sub MakeLoanTable()
   Dim XLSheet As Object
```

```
   Dim LoanAmt
   Dim Wbook As String
'  Запрос значений
   LoanAmt = InputBox("Сумма кредита?")
   If LoanAmt = "" Then Exit Sub

'  Очистка документа
   ThisDocument.Content.Delete

'  Создание объекта Sheet
   Wbook = ThisDocument.Path & "\mortgagecalcs.xlsx"
   Set XLSheet = GetObject(Wbook, "Excel.Sheet").ActiveSheet

'  Передача значений в лист
   XLSheet.Range("LoanAmount") = LoanAmtXLSheet.Calculate
'  Вставка заголовка страницы
   Selection.Style = "Заголовок"
   Selection.TypeText "Величина кредита: " & _
      Format(LoanAmt, "$#,##0")
   Selection.TypeParagraph
   Selection.TypeParagraph
'  Копирование данных с листа и их вставка в документ
   XLSheet.Range("DataTable").Copy
   Selection.Paste

   Selection.TypeParagraph
   Selection.TypeParagraph

'  Копирование и вставка в документ диаграммы
   XLSheet.ChartObjects(1).Copy
   Selection.PasteSpecial _
      Link:=False, _
      DataType:=wdPasteMetafilePicture, _
      Placement:=wdInLine
'  Уничтожение объекта
   Set XLSheet = Nothing
End Sub
```

Компакт-диск

Рассматриваемый в этом разделе пример находится на прилагаемом к книге компакт-диске. Документ Word находится в файле `automate excel.docm`, а рабочая книга Excel — в файле `mortgagecalcs.xlsx`. После открытия документа Word вызовите на выполнение макрос `MakeLoanTable`, выполнив команду Вставка⇨Mortgage⇨Get Mortgage Amount (Вставка⇨Ипотека⇨Получить размер ипотеки).

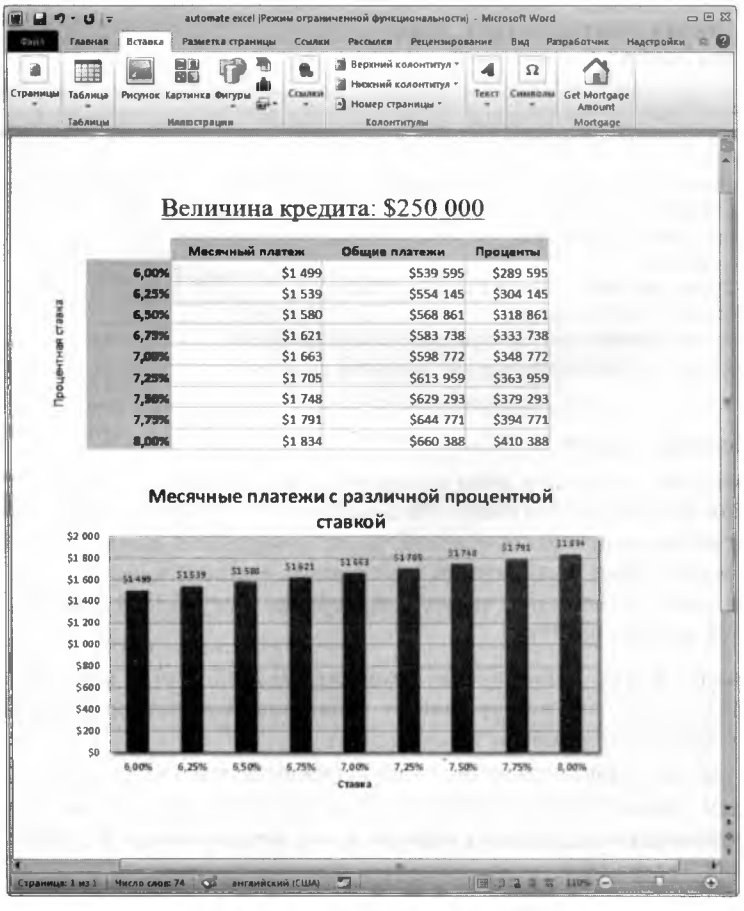
Рабочая книга Excel, используемая этой процедурой Word, показана на рис. 20.8. Процедура `MakeLoanTable` запрашивает у пользователя размер кредита и вставляет значение в ячейку C7 (под именем `LoanAmount`).

Пересчет рабочего листа приводит к обновлению таблицы данных в диапазоне F2:I12 (под названием `DataTable`), а также обновляет диаграмму. Затем диапазон `DataTable` и диаграмма копируются из объекта Excel и вставляются в документ Word. Результат показан на рис. 20.9.

Параметры кредита	
Сумма кредита:	$400 000
Процентная ставка:	6,50%
Срок кредита:	360
Месячный платеж:	$2 528
Общие платежи:	$910 178
Общие платежи по процентам:	$510 178

Процентная ставка	Месячный платеж	Общие платежи	Проценты
6,00%	$2 398	$863 353	$463 353
6,25%	$2 463	$886 633	$486 633
6,50%	$2 528	$910 178	$510 178
6,75%	$2 594	$933 981	$533 981
7,00%	$2 661	$958 036	$558 036
7,25%	$2 729	$982 334	$582 334
7,50%	$2 797	$1 006 869	$606 869
7,75%	$2 866	$1 031 634	$631 634
8,00%	$2 935	$1 056 621	$656 621

Рис. 20.8. Созданная в Word процедура VBA использует этот рабочий лист



Величина кредита: $250 000

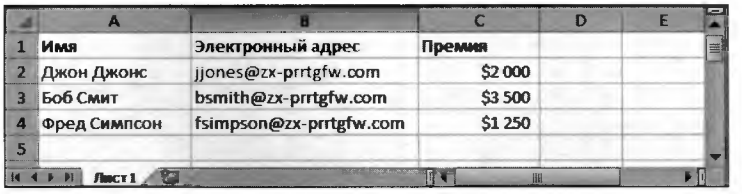
Процентная ставка	Месячный платеж	Общие платежи	Проценты
6,00%	$1 499	$539 595	$289 595
6,25%	$1 539	$554 145	$304 145
6,50%	$1 580	$568 861	$318 861
6,75%	$1 621	$583 738	$333 738
7,00%	$1 663	$598 772	$348 772
7,25%	$1 705	$613 959	$363 959
7,50%	$1 748	$629 293	$379 293
7,75%	$1 791	$644 771	$394 771
8,00%	$1 834	$660 388	$410 388

Рис. 20.9. Процедура VBA для Word использует Excel для создания данного документа

Отправка почтовых сообщений с помощью Outlook

В этом разделе иллюстрируются методы автоматизации с программой Microsoft Outlook.

На рис. 20.10 показан рабочий лист, который содержит данные почтового сообщения: имя, почтовый адрес и сумму предоставляемого пособия. Процедура `SendMail` просматривает строки (записи) этой рабочей книги и создает на основе полученных данных отдельные сообщения (сохраненные в переменной `Msg`).



	A	B	C	D	E
1	Имя	Электронный адрес	Премия		
2	Джон Джонс	jjones@zx-prrtgfw.com	$2 000		
3	Боб Смит	bsmith@zx-prrtgfw.com	$3 500		
4	Фред Симпсон	fsimpson@zx-prrtgfw.com	$1 250		
5					

Рис. 20.10. Эта информация используется почтовым клиентом Outlook для создания сообщений

```
Sub SendEmail()
' Использует раннее связывание.
' Требует ссылку на библиотеку объектов Outlook
  Dim OutlookApp As Outlook.Application
  Dim MItem As Outlook.MailItem
  Dim cell As Range
  Dim Subj As String
  Dim EmailAddr As String
  Dim Recipient As String
  Dim Bonus As String
  Dim Msg As String

' Создание объекта Outlook
  Set OutlookApp = New Outlook.Application

' Циклический просмотр столбцов
  For Each cell In Columns("B").
     Cells.SpecialCells(xlCellTypeConstants)
     If cell.Value Like "*@*" Then
'       Получение данных
        Subj = "Ваша премия"
        Recipient = cell.Offset(0, -1).Value
        EmailAddr = cell.Value
        Bonus = Format(cell.Offset(0, 1).Value, "$0,000.")

'       Составление сообщения
        Msg = "Дорогой " & Recipient & vbCrLf & vbCrLf
        Msg = Msg & "Рад сообщить вам о начислении премии "
        Msg = Msg & Bonus & vbCrLf & vbCrLf
        Msg = Msg & "Вильям Роуз" & vbCrLf
        Msg = Msg & "Президент"

'       Создание элемента сообщения и его отправка
        Set MItem = OutlookApp.CreateItem(olMailItem)
        With Mitem
           .To = EmailAddr
```

```
            .Subject = Subj
            .Body = Msg
            .Send
        End With
    End If
  Next
End Sub
```

На рис. 20.11 показано одно из электронных сообщений, отображаемых в окне Outlook.

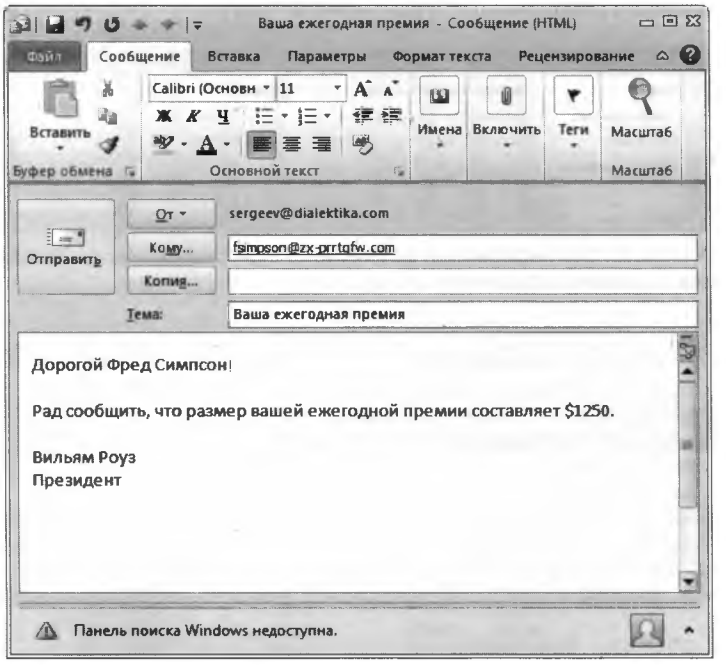


Рис. 20.11. Окно Outlook с почтовым сообщением, созданным Excel

В этом примере демонстрируется раннее связывание, поэтому в нем применяются ссылки на библиотеку объектов Outlook. Обратите внимание, что используются два объекта: `Outlook` и `MailItem`. Первый был создан с помощью следующего оператора:

```
Set OutlookApp = New Outlook.Application
```

Объект `MailItem` был создан с помощью следующего оператора.

```
Set MItem = OutlookApp.CreateItem(olMailItem)
```

Код устанавливает значения свойств `To`, `Subject` и `Body`, а также использует метод `Send` для отправки сообщений.

Совет

Для сохранения сообщений в папке `Черновики` (Draft) воспользуйтесь методом `Save` вместо метода `Send`. Это особенно полезно в случае тестирования и отладки кода.

Если только вы не изменили настройки системы безопасности, то при отправке каждого сообщения на экране будет появляться диалоговое окно, показанное на рис. 20.12. Для устранения этого сообщения активизируйте Outlook и выберите команды **Файл⇨**

Параметры Outlook⇨Центр управления безопасностью (Office⇨Outlook Options⇨ Trust Center). Затем щелкните на кнопке Параметры центра управления безопасностью (Trust Center Settings). В окне Параметры центра управления безопасностью (Trust Center) перейдите в раздел Программный доступ (Programmatic Access) и установите переключатель Никогда не предупреждать о подозрительной активности (не рекомендуется) (Never Warn Me about Suspicious Activity (Not Recommended)). Помните, что при этом возникает угроза безопасности вашей системы.

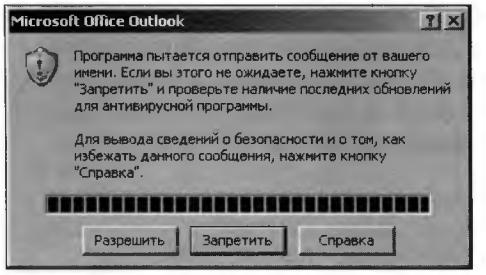


Рис. 20.12. При использовании Excel для отправки электронных сообщений Outlook обычно отображается предупреждающее сообщение

Компакт-диск

Пример этого раздела находится на прилагаемом к книге компакт-диске в файле `personalized email-outlook.xlsm`. Для работы с этим примером нужно установить Microsoft Outlook. На компакт-диске также находится несколько измененная версия макроса, в которой используется позднее связывание. `personalized email-outlook (late binding).xlsm`.

Примечание

В следующих разделах описываются другие методы отправки почтовых сообщений с помощью Excel.

Отправка почтовых вложений с помощью Excel

Как вы наверняка знаете, в Excel включены команды для отправки рабочих листов и рабочих книг в виде вложений в почтовые сообщения. Конечно, с помощью VBA вы можете автоматизировать подобные операции. Приведенная ниже процедура отправляет активную рабочую книгу (как вложение) по адресу `joeblow@zx-prrtgfw.com`. Почтовое сообщение имеет тему `Моя книга`.

```
Sub SendWorkbook()
   ActiveWorkbook.SendMail "joeblow@zx-prrtgfw.com", "Моя книга"
End Sub
```

Примечание

Метод `SendMail` работает с почтовым клиентом, заданным по умолчанию.

Если требуется отправить только отдельный рабочий лист книги, то вам придется скопировать его в новую (временную) рабочую книгу. После отправки сообщения в виде

вложения удалите временный файл. Ниже приведен пример отправки листа Лист1 активной рабочей книги.

```
Sub Sendasheet()
   ActiveWorkbook.Worksheets("Лист1").Copy
   ActiveWorkbook.SendMail "joeblow@zx-prrtgfw.com", "Мой лист"
   ActiveWorkbook.Close False
End Sub
```

В предыдущем примере файлу было присвоено имя рабочей книги по умолчанию (например, Книга2.xlsx). Для того чтобы рабочей книге, включающей один рабочий лист и пересылаемой во вложении, присвоить более значимое имя, придется сохранить файл на диске и удалить его после отправки. В следующей процедуре лист Лист1 сохраняется в файле Мой файл.xlsx. После отправки временная рабочая книга в виде вложения удаляется с помощью оператора VBA Kill.

```
Sub SendOneSheet()
   Dim Filename As String
   Filename = "Мой файл.xlsx"
   ActiveWorkbook.Worksheets("Лист1").Copy
   ActiveWorkbook.SaveAs Filename
   ActiveWorkbook.SendMail "joeblow@zx-prrtgfw.com", "Мой лист"
   ActiveWorkbook.Close False
   Kill Filename
End Sub
```

Примечание

К сожалению, в Excel отсутствует способ автоматического сохранения рабочей книги в виде PDF-файла с последующей отправкой этого файла в виде почтового вложения. Однако в ваших силах автоматизировать часть этого процесса. Ниже представлен код процедуры SendSheetAsPDF, которая сохраняет активный лист в виде PDF-файла, а затем отображает окно составления сообщения почтового клиента, заданного по умолчанию (с вложенным PDF-файлом). Пользователю остается указать имя получателя и щелкнуть на кнопке Отправить (Send).

```
Sub SendSheetAsPDF()
   CommandBars.ExecuteMso ("FileEmailAsPdfEmailAttachment")
End Sub
```

Когда Excel "выбивается из сил", на сцену "выходит" Outlook. Следующая процедура сохраняет активную рабочую книгу в виде PDF-файла, а также автоматизирует Outlook таким образом, что создается электронное сообщение с вложенным PDF-файлом.

```
Sub SendAsPDF()
'  Используется раннее связывание
'  Требуется ссылка на объектную библиотеку Outlook
   Dim OutlookApp As Outlook.Application
   Dim MItem As Object
   Dim Recipient As String, Subj As String
   Dim Msg As String, Fname As String

'  Подробности сообщения
   Recipient = "myboss@xrediyh.com"
   Subj = "Отчет по продажам"
   Msg = "Босс, вот требуемый PDF-файл."
   Msg = Msg & vbNewLine & vbNewLine & "-Франк"
   Fname = Application.DefaultFilePath & "\" & _
```

```
    ActiveWorkbook.Name & ".pdf"

  Создание почтового вложения
  ActiveSheet.ExportAsFixedFormat _
    Type:=xlTypePDF, _
    Filename:=Fname

  Создание объекта Outlook
  Set OutlookApp = New Outlook.Application

' Создание почтового вложения и его отправка
  Set MItem = OutlookApp.CreateItem(olMailItem)
  With Mitem
    .To = Recipient
    .Subject = Subj
    .Body = Msg
    .Attachments.Add Fname
    .Save ' В папке Черновики (Drafts)
    ' .Send
  End With
  Set OutlookApp = Nothing
' Удаление файла
  Kill Fname
End Sub
```

Компакт-диск

Пример из этого раздела находится на прилагаемом к книге компакт-диске в файле `send pdf via outlook.xlsm`.

Использование метода SendKeys

Не все приложения поддерживают автоматизацию. В некоторых случаях можно управлять такими приложениями. Для этого воспользуйтесь методом Excel `SendKeys`, который в результате отправки комбинаций клавиш эмулирует действия пользователя.

Несмотря на то что использование метода `SendKeys` является довольно удачным решением, его применение связано с рядом трудностей. Потенциальная проблема заключается в том, что метод применяется по отношению к конкретному пользовательскому интерфейсу. Если более поздняя версия программы имеет другой интерфейс, то приложение, использующее метод `SendKeys`, перестанет работать. Следовательно, применять метод `SendKeys` можно только в качестве последнего средства.

В следующем примере демонстрируется применение метода `SendKeys` к программе Windows Calculator (Калькулятор). Процедура `TestKeys` переводит программу в инженерный режим (команда **Вид⇨Инженерный** (View⇨Scientific)).

```
Sub TestKeys()
  Shell "calc.Exe", vbNormalFocus
  Application.SendKeys "%vs"
End Sub
```

В данном примере отправляется комбинация клавиш <Alt+V> (символ процента представляет клавишу <Alt>), за которой следует клавиша <S>. Документация по использованию метода `SendKeys` представлена в справочной системе, в которой описаны вопросы отправки нестандартных комбинаций клавиш (например, комбинаций с клавишей <Alt> или <Ctrl>).

Примечание

В качестве последнего упражнения этой главы я попытался запустить процедуру `TestKeys` на компьютере с установленной операционной системой Windows 7. Несмотря на то что стандартный калькулятор в Windows 7 использует те же клавиши — ускорители меню, что и калькулятор в Windows XP, процедура не работает. На основании этого небольшого исследования я пришел к выводу о том, что Windows 7 поддерживает метод `SendKeys`, только если отключен контроль учетных записей (свойство обеспечения безопасности). Это еще один яркий пример того, что к методу `SendKeys` следует прибегать в крайнем случае.

Глава 21

Создание и использование надстроек

В этой главе...

- Определение надстройки
- Использование диспетчера надстроек Excel
- Создание надстройки
- Пример надстройки
- Сравнение файлов XLAM и XLSM
- Управление надстройками с помощью кода VBA
- Оптимизация производительности надстроек
- Проблемы, связанные с использованием надстроек

В этой главе рассматриваются преимущества использования надстроек. Речь пойдет о создании надстроек с помощью инструментов, встроенных в Excel.

Определение надстройки

Одна из самых полезных возможностей Excel, которой пользуются разработчики, — это создание надстроек. Именно надстройки являются подтверждением высокого профессионализма разработчика, к тому же они имеют массу преимуществ по сравнению со стандартными файлами рабочих книг.

Вообще говоря, *надстройка электронной таблицы* — это средство, добавляемое к листу для обеспечения дополнительной функциональности. Например, Excel поставляется с несколькими заранее созданными надстройками. Одной из самых популярных является надстройка *Пакет анализа* (Analysis ToolPak). Она добавляет в программу инструменты анализа и статистической обработки данных, которые по умолчанию в Excel отсутствуют.

Некоторые надстройки также предоставляют новые функции рабочих листов, которые можно добавлять в создаваемые формулы. Новые средства обычно эффективно дополняют исходный интерфейс, а потому становятся неотъемлемой частью программы.

Сравнение надстройки со стандартной рабочей книгой

Любой опытный пользователь Excel может создать надстройку на основе рабочей книги Excel. Для этого не потребуются дополнительное программное обеспечение и специальные средства. Любой файл рабочей книги можно преобразовать в надстройку, но не все файлы подходят на роль надстройки. В Excel надстройка представлена обычной рабочей книгой XLSM (за небольшим исключением).

- Свойству `IsAddin` объекта `ThisWorkbook` (ЭтаКнига) присвоено значение `True`. По умолчанию этому свойству присваивается значение `False`.
- Окно рабочей книги скрыто так, что его нельзя отобразить с помощью команды Вид⇨Окно⇨Отобразить окно (View⇨Window⇨Unhide). Это означает невозможность отображения листов рабочей книги либо диаграмм до тех пор, пока не будет создан макрос, который копирует лист в стандартную рабочую книгу.
- Надстройка не входит в коллекцию `Workbooks` — она является составной частью коллекции `AddIns`. Тем не менее можно получить доступ к надстройке с помощью коллекции `Workbooks` (см. раздел "Членство в коллекциях").
- Надстройки можно инсталлировать и деинсталлировать с помощью диалогового окна Надстройки (Add-Ins). Для его отображения выполните команду Файл⇨Параметры Excel⇨Надстройки (File⇨Excel Options⇨Add-Ins). В диалоговом окне Параметры Excel (Excel Options) в списке Управление (Manage) выберите пункт Надстройки Excel (Excel Add-Ins) и щелкните на кнопке Перейти (Go). После завершения установки надстройка появится при следующем запуске Excel.
- Названия макросов, находящихся в надстройках, не отображаются в диалоговом окне Макрос (Macro). Для перехода в это окно выполните команду Разработчик⇨Код⇨Макросы (Developer⇨Code⇨Macros) или Вид⇨Макросы⇨Макросы (View⇨Macros⇨Macros).
- Функции рабочего листа, хранящиеся в надстройках, могут применяться в формулах без указания названия файла исходной рабочей книги.

Примечание

Ранее в Excel разрешалось применение любого расширения для файла надстройки. Начиная с версии Excel 2007 эта возможность сохранилась, но теперь, если не применяется расширение XLA или XLAM, на экране отображается предупреждающее сообщение (рис. 21.1). Оно появляется даже в том случае, когда установленная надстройка открывается автоматически при запуске Excel.

Рис. 21.1. Excel отображает предупреждение, если файл надстройки имеет нестандартное расширение

Основные причины создания надстроек

Ниже приведены причины, которые могут заставить пользователя конвертировать приложение Excel в надстройку.

- **Для ограничения доступа к коду и рабочим листам.** Когда приложение распространяется в виде надстройки, оно защищается с помощью пароля. Чтобы пользователи не смогли получить доступ к листам и модифицировать конфиденциальные методы, следует предотвратить копирование кода или как минимум усложнить такую задачу.
- **Для внесения ясности.** Если пользователь загружает приложение в виде надстройки, то файл остается невидимым, поэтому вероятность того, что начинающий пользователь на него наткнется, намного снижается. В отличие от скрытых рабочих книг, надстройку невозможно отобразить.
- **Для упрощения доступа к функциям рабочего листа.** Создаваемые пользователем функции рабочего листа, заданные в надстройке, не требуют указания имени рабочей книги перед вызовом. Например, если в рабочей книге `Newfuncs` содержится функция MOVAVG, то для ее использования в другой рабочей книге необходимо обратиться к следующему синтаксису:

  ```
  =Newfuncs.xlsm!MOVAVG(A1:A50)
  ```

  Если функция будет храниться в открытой надстройке, то можно использовать более простой синтаксис, так как отсутствует необходимость добавления ссылки на файл.

  ```
  =MOVAVG(A1:A50)
  ```

- **Для упрощения доступа пользователям.** Как только расположение надстройки будет указано, она отобразится в диалоговом окне Надстройки (Add-Ins) с дружественным именем и описанием возможностей.
- **Для получения дополнительного контроля над процессом загрузки.** Надстройки могут открываться автоматически при запуске Excel независимо от папки, в которой они расположены.
- **Чтобы избежать отображения сообщений при выгрузке**. При закрытии надстройки не отображаются сообщения `Сохранить изменения в` *`xxx`*`?`.

Примечание

Возможность применения надстроек определяется пользовательскими настройками безопасности, установленными в диалоговом окне Центр управления безопасностью (Trust Center). Для отображения этого окна выполните команду Разработчик⇨Код⇨Безопасность макросов (Developer⇨Macro⇨Security). Если же вкладка Разработчик (Developer) не отображается, выполните команду Файл⇨Параметры Excel⇨Центр управления безопасностью (File⇨Excel Options⇨Trust Center) и щелкните на кнопке Параметры центра управления безопасностью (Trust Center Settings).

О надстройках COM

В Excel также поддерживаются надстройки COM (Component Object Model). Файлы этих надстроек имеют расширение `.dll` или `.exe`. Надстройки COM сохраняются таким образом, что могут работать со всеми приложениями Office, которые поддерживают такую возможность. Дополнительным преимуществом является компилируемость кода, что увеличивает безопасность. В отличие от надстроек XLAM, надстройка COM не

может включать листы или диаграммы Excel. Для разработки надстроек COM используются Visual Basic 5 (или более поздней версии), а также Visual Basic .NET. Вопросы создания надстроек COM выходят за рамки этой книги.

Использование диспетчера надстроек Excel

Наиболее эффективный способ управления надстройками обеспечивает диалоговое окно Надстройки (Add-Ins), которое появляется после выполнения команды Файл⇨Параметры Excel⇨Надстройки (File⇨Excel Options⇨Add-Ins). Затем в диалоговом окне Параметры Excel (Excel Options) в списке Управление (Manage) выберите пункт Надстройки Excel (Excel Add-Ins) и щелкните на кнопке Перейти (Go).

Совет

В прежних версиях Excel для быстрого перехода к окну Надстройки (Add-Ins) использовалась комбинация клавиш <Alt+TI>. В Excel 2010 также появилась группа команд Надстройки (Add-Ins), находящаяся на ленте Разработчик (Developer).

Диалоговое окно Параметры Excel (Add-Ins) показано на рис. 21.2. В списке перечислены названия всех установленных надстроек, а флажки идентифицируют все открытые надстройки. Открывать или закрывать надстройки в этом диалоговом окне можно с помощью флажков.

Рис. 21.2. Диалоговое окно Надстройки

Предупреждение

Большинство файлов надстроек можно открыть с помощью команды Файл⇨Открыть (File⇨Open). Поскольку надстройка не является активной рабочей книгой, ее невозможно закрыть с помощью команды Файл⇨Закрыть (File⇨Close). Для удаления надстройки следует выйти из Excel и повторно запустить эту программу либо выполнить макрос VBA, закрывающий надстройку, как показано ниже.

```
Workbooks("myaddin.xlam").Close
```

При открытии надстройки с помощью команды Файл⇨Открыть (File⇨Open) открывается сам файл, а надстройка не устанавливается.

После открытия файла надстройки нетрудно заметить изменения в "облике" Excel. Как правило, изменяется пользовательский интерфейс: на ленте появляется новая команда либо в контекстном меню появляется новый пункт. Например, после установки надстройки Пакет анализа (Analysis ToolPak) на ленте появляется новая команда: Данные⇨Анализ⇨Пакет анализа (Data⇨Analysis⇨Data Analysis). После установки надстройки Excel Инструменты для евро (Euro Currency Tools) во вкладке Формулы (Formulas) появляется новая группа — Решения (Solutions).

Если надстройка содержит только пользовательские функции рабочего листа, эти функции появляются в диалоговом окне Вставка функции (Insert Function) и в группе ленты Библиотека функций (Function Library).

Примечание

При открытии надстроек, созданных в версиях до Excel 2007, изменения в пользовательском интерфейсе могут не отображаться в том виде, в каком они должны появляться. В этом случае для получения доступа к элементам интерфейса, создаваемых с помощью надстройки (меню и панели инструментов), выберите пункты меню Надстройки⇨Команды меню (Add-Ins⇨Menu Commands) или Надстройки⇨Настраиваемые панели инструментов (Add-Ins⇨Custom Toolbars).

Создание надстройки

Как уже отмечалось, можно преобразовать рабочую книгу в надстройку, хотя далеко не все рабочие книги для этого подходят. Надстройка должна включать макрос (иначе она бесполезна).

Больше всего пользы принесет преобразование в надстройку рабочей книги, содержащей универсальные макросы. Если же в книге, кроме листов, ничего нет, созданная на ее основе надстройка будет невидимой, а следовательно, недоступной для пользователя. Конечно, можно написать макрос, который будет копировать листы из надстройки в рабочую книгу, открывая доступ к ним со стороны пользователя.

Создание надстройки на основе рабочей книги не представляет особого труда. Просто выполните следующие действия.

1. Разработайте приложение и убедитесь в том, что оно работает корректно.
2. Не забудьте предусмотреть способ вызова на выполнение макроса (или макросов) в надстройке. Для этого можете добавить новую команду на ленту или новый пункт контекстного меню. Дополнительные сведения об изменении пользовательского интерфейса Excel можно найти в главах 22 и 23.
3. Активизируйте редактор Visual Basic Editor (VBE) и выберите рабочую книгу в окне Project (Проект).
4. Выберите команду Tools⇨*xxx* Properties (Сервис⇨*xxx* Свойства), где *xxx* — это название проекта. Щелкните на кнопке Protection (Защита). Установите флажок Lock Project for Viewing (Заблокировать просмотр проекта) и дважды введите пароль. Щелкните на кнопке OK.

Это действие обязательно для выполнения в том случае, когда нужно предотвратить просмотр или изменение макроса или диалогового окна `UserForm`.

5. Повторно запустите Excel и выполните команду **Разработчик⇨Изменение⇨Область документа** (Developer⇨Modify⇨Document Panel) для отображения панели **Область сведения о документе** (Document Properties).
6. Введите короткий описательный заголовок в поле **Название** (Title) и более длинное описание в поле **Примечания** (Comments).

   Это действие необязательно, хотя и облегчает дальнейшее использование надстройки путем отображения описательного текста в диалоговом окне **Надстройки** (Add-Ins).
7. Выберите команду **Файл⇨Сохранить как** (File⇨Save As). Отобразится диалоговое окно **Сохранение документа** (Save As).
8. В диалоговом окне **Сохранение документа** (Save As) в раскрывающемся списке **Тип файла** (Save as Type) выберите пункт **Надстройка Excel** (Add-In (*.xlam)).
9. Щелкните на кнопке **Сохранить** (Save). Будет сохранена копия рабочей книги (с расширением `.xlam`), а исходная книга останется открытой.
10. Закройте исходную рабочую книгу и установите созданную на ее основе надстройку.
11. Протестируйте надстройку, чтобы убедиться в ее работоспособности.

    Если надстройка не работает, внесите изменения в код макроса. И не забудьте сохранить изменения.

Предупреждение

Рабочая книга, преобразованная в надстройку, должна включать хотя бы один лист. Например, если рабочая книга содержит только один лист диаграммы или лист диалога Excel 5/95, пункт Надстройка Excel исчезнет из раскрывающегося списка в окне Сохранение документа. Также этот пункт отображается в списке только в том случае, когда рабочий лист активен в момент выбора команды Файл⇨Сохранить как (File⇨Save As).

Пример надстройки

В этом разделе будет рассмотрен пример создания полезной надстройки. В процессе ее создания используется утилита, выполняющая экспорт диаграмм в отдельные графические файлы. Эта утилита добавляет новую группу в контекстную вкладку **Работа с диаграммами⇨Конструктор** (Chart Tools⇨Design). Главное диалоговое окно этой утилиты показано на рис. 21.3. Эта утилита достаточно сложная и, чтобы разобраться в принципах ее работы, понадобится время.

Компакт-диск

Версия `XLSM` утилиты Export Charts (под именем `export charts.xlsm`) находится на прилагаемом к книге компакт-диске. На основе этого файла можно создать описываемую надстройку.

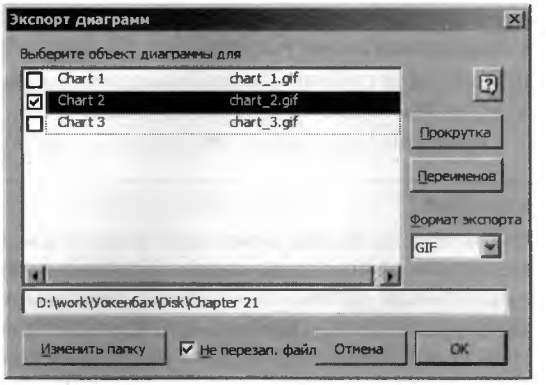


Рис. 21.3. На основе рабочей книги `Export Charts` создается полезная надстройка

В этом примере мы будем работать с рабочей книгой, которая была создана и отлажена ранее. Эта книга включает следующие элементы.

- **Рабочий лист `Лист1`**. На этом листе находятся исходные данные, которые могут применяться вместо измененных данных в случае выбора пользователем операции отмены.
- **Диалоговое окно `UserForm` под названием `UserForm1`**. Это диалоговое окно применяется в качестве основного пользовательского интерфейса. Модуль кода для этого окна включает несколько процедур обработки событий.
- **Диалоговое окно `UserForm` под названием `UserForm2`.** Это диалоговое окно отображается после щелчка на кнопке Переименовать (Rename) и позволяет изменить имя файла экспортируемой диаграммы.
- **Диалоговое окно `UserForm` под названием `UserForm3`.** Это диалоговое окно отображается в случае открытия рабочей книги. В нем отображается краткая инструкция по получения доступа к утилите Export Charts. Также отображается флажок Не показывать сообщение снова (Don't Show This Message Again).
- **Модуль VBA под названием `Module1`.** Этот модуль содержит ряд процедур, включая процедуру, которая отображает диалоговое окно `UserForm` под названием `UserForm1`.
- **Модуль кода `ThisWorkbook` (ЭтаКнига).** Этот модуль содержит процедуру `Workbook_Open`, которая считывает сохраненные настройки и отображает приветственное сообщение.
- **XML-код, предназначенный для настройки ленты.** Подобная настройка должна выполняться вне Excel. Обратитесь к главе 22 для получения дополнительных сведений о настройке ленты с помощью RibbonX.

Добавление описания в надстройку

Для добавления названия и описания надстройки выберите команду Разработчик⇨Изменение⇨Область документа (Developer⇨Modify⇨Document Panel). На экране появится панель Свойства документа (Document Properties), которая отображается под лентой.

В поле Название (Title) введите название надстройки. Текст названия отображается в списке надстроек в диалоговом окне Надстройки (Add-Ins). В поле Примечания (Comments) введите описание надстройки. Эта информация отображается в нижней части диалогового окна Надстройки после выбора надстройки.

Указывать название и описание надстройки необязательно, но крайне желательно.

Несколько слов о паролях

Компания Microsoft никогда не позиционировала Excel в качестве средства разработки приложений с защищенным исходным кодом. Используемый в Excel пароль предотвращает доступ к отдельным частям приложения неискушенным пользователям. В Excel 2002 и более поздних версиях уровень безопасности был усилен, хотя пароль может быть взломан опытными хакерами. И если вы хотите исключить возможность доступа к макросам или формулам, вряд ли Excel будет наилучшим выбором в качестве платформы разработки.

Формирование надстройки

Для создания надстройки выполните следующие действия.

1. Активизируйте VBE и выберите рабочую книгу, на основе которой будет создана надстройка, в окне Project (Проект).
2. Выберите команду Debug⇨Compile (Отладка⇨Компиляция).

   После этого начнется компиляция кода VBA, в процессе которой выявляются синтаксические ошибки, которые затем исправляются пользователем. Если сохранить рабочую книгу в качестве надстройки, Excel создаст надстройку, даже если в ней есть синтаксические ошибки.
3. Выберите команду Tools⇨*xxx* Properties (Сервис⇨*xxx* Свойства), где *xxx* — это название проекта, для отображения диалогового окна Project Properties (Свойства проекта). Щелкните на вкладке General (Общие) и введите новое имя проекта.

   По умолчанию все проекты VB называются `VBProject`. В данном примере название проекта изменяется на `ExpCharts`. Этот шаг дополнительный, хотя и рекомендуемый.
4. Сохраните рабочую книгу в файле с именем `*.XLSM`. Вообще говоря, этот шаг необязателен, но он нужен для создания резервной копии XLSM (без пароля) файла надстройки XLAM.
5. В диалоговом окне Project Properties (Свойства проекта) выберите вкладку Protection (Защита). Установите флажок Lock Project for Viewing (Заблокировать просмотр проекта) и дважды введите пароль.

   После этого код по-прежнему будет отображаться на экране, а изменения скажутся после закрытия и повторного открытия файла.
6. Щелкните на кнопке OK.

   Если вы не хотите защищать проект, пропустите пп. 5 и 6.
7. В Excel выберите команду Файл⇨Сохранить как (File⇨Save As). После этого Excel отобразит окно Сохранение документа (Save As).

8. В раскрывающемся списке Тип файла (Save as Type) выберите пункт Надстройка Excel (Excel Add-In (*.xlam)).
9. Щелкните на кнопке Сохранить (Save).

   После этого будет создана надстройка, причем исходный файл (XLSM-версия) останется открытым.

Надстройки могут размещаться в любой папке.

Диспетчер надстроек Excel

Установить и отменить установку надстроек можно с помощью диалогового окна Надстройки (Add-Ins) в Excel. Для отображения этого окна выберите команду Файл⇨Параметры Excel ⇨Надстройки (File⇨Excel Options⇨Add-Ins). В диалоговом окне Параметры Excel (Excel Options) в списке Управление (Manage) выберите пункт Надстройки Excel (Excel Add-Ins) и щелкните на кнопке Перейти (Go). В этом диалоговом окне отображаются названия доступных надстроек. Все отмеченные флажками надстройки открыты.

Говоря языком программистов VBA, в диалоговом окне Надстройки (Add-Ins) перечисляются значения свойства `Title` каждого объекта `AddIn` в коллекции `AddIns`. Каждая надстройка, для которой установлен флажок, — это свойство `Installed`, которому присвоено значение `True`.

Для установки надстройки следует установить соответствующий флажок. Если же нужно деинсталлировать надстройку, отмените установку соответствующего ей флажка. Для добавления надстройки в список воспользуйтесь кнопкой Просмотр (Browse), с помощью которой найдите требуемый файл. По умолчанию в диалоговом окне Надстройки (Add-Ins) отображаются надстройки следующих типов:

- **XLAM** — надстройка Excel 2007, созданная на основе файла `XLSM`;
- **XLA** — надстройка версии Excel, более ранней, чем Excel 2007, созданной на основе XLS-файла;
- **XLL** — отдельный скомпилированный файл DLL.

Если щелкнуть на кнопке Автоматизация (Automation), которая появилась в Excel 2002 и более поздних версиях, в открывшемся окне можно выбрать надстройки COM. Обратите внимание, что в диалоговом окне Серверы автоматизации (Automation Servers) может отображаться очень много файлов, и не все из них являются надстройками COM, которые могут работать в Excel.

Надстройку можно зарегистрировать в коллекции `AddIns` с помощью метода `Add` из коллекции VBA `AddIns`, но нельзя удалить с помощью кода VBA. Можно также открыть надстройку с помощью кода VBA, присвоив свойству `Installed` объекта `AddIn` значение `True`. Если этому свойству присваивается значение `False`, надстройка закрывается.

Диспетчер надстроек сохраняет состояние установленных надстроек в реестре Windows после выхода из Excel. Таким образом, все установленные надстройки будут автоматически открыты после повторного запуска Excel.

Установка надстройки

Во избежание путаницы закрывайте рабочую книгу `XLSM` после установки созданной на ее основе надстройки.

Для установки надстройки выполните следующие действия.

1. Выберите команду Файл⇨Параметры Excel⇨Надстройки (File⇨Excel Options⇨Add-Ins).
2. В диалоговом окне Параметры Excel (Excel Options) из раскрывающегося списка Управление (Manage) выберите пункт Параметры Excel (Excel Add-Ins) и щелкните на кнопке Перейти (Go) либо нажмите <Alt+TI>.

   После этого Excel отобразит диалоговое окно Надстройки (Add-Ins).
3. Щелкните на кнопке Просмотр (Browse), найдите только что созданную надстройку и дважды щелкните на ней мышью.

   После нахождения файла новой надстройки сама надстройка будет отображаться в списке диалогового окна Надстройки (Add-Ins). Как показано на рис. 21.4, в диалоговом окне Надстройки (Add-Ins) также отображается описательная информация, которая была определена в окне Properties.
4. Щелкните на кнопке OK для закрытия диалогового окна и открытия надстройки.

После открытия надстройки Export Charts на контекстной вкладке Работа с диаграммами⇨Конструктор (Chart Tools⇨Design) появится новая группа Export Charts (Экспорт диаграмм) с двумя кнопками. После щелчка на первой кнопке отображается диалоговое окно Экспорт диаграмм (Export Charts), а после щелчка на второй — содержимое файла справки. Обратите внимание, что контекстная вкладка Работа с диаграммами⇨Конструктор отображается только в том случае, если выбрана диаграмма (или лист диаграммы).

Тестирование надстройки

После установки надстройки следует провести ее тестирование. Откройте новую рабочую книгу и протестируйте различные свойства утилиты Export Charts. Выполняйте различные тесты, чтобы выяснить возможные проблемы. А еще лучше — обратитесь к человеку, не знакомому с приложением, который сможет провести настоящий "краш-тест".

При выявлении каких-либо ошибок исправьте код надстройки (исходный файл при этом не нужен). После выполнения изменений сохраните файл, выбрав команду File⇨Save (Файл⇨Сохранить) редактора VBE.

Распространение надстройки

Для распространения надстроек достаточно просто передать "заинтересованным лицам" копию файла XLAM (версия XLSM этого же файла не нужна) с инструкциями по его установке. Если файл заблокировать паролем, доступ к коду макроса откроется только после ввода пароля.

Изменение надстройки

Если нужно изменить надстройку, сначала откройте ее, а затем разблокируйте VB-проект, защищенный паролем. Для разблокирования активизируйте редактор VBE и дважды щелкните мышью на имени проекта в окне Project (Проект). При этом может отобразиться запрос о вводе пароля. Выполните необходимые изменения и сохраните файл в VBE (с помощью команды File⇨Save (Файл⇨Сохранить)).

Если создается надстройка, которая сохраняет информацию в рабочем листе, то для просмотра рабочей книги в Excel свойству `IsAddIn` нужно присвоить значение `False`.

Для этого следует выбрать объект `ThisWorkbook` (ЭтаКнига) и перейти в окно Properties (Свойства) для этого объекта (рис. 21.5). После выполнения необходимых изменений присвойте свойству `IsAddIn` значение `True` перед сохранением файла. Если значение свойства `IsAddIn` будет `False`, Excel не позволит сохранить файл с расширением XLAM.

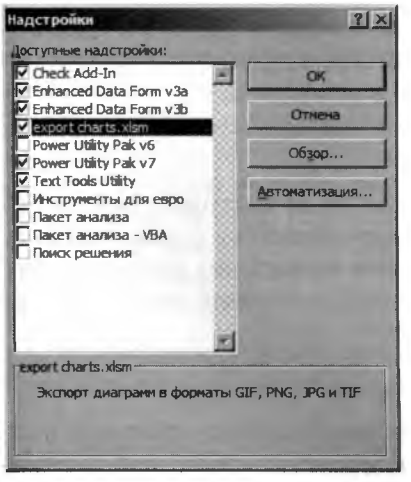


Рис. 21.4. В диалоговом окне Надстройки отображается установленная надстройка

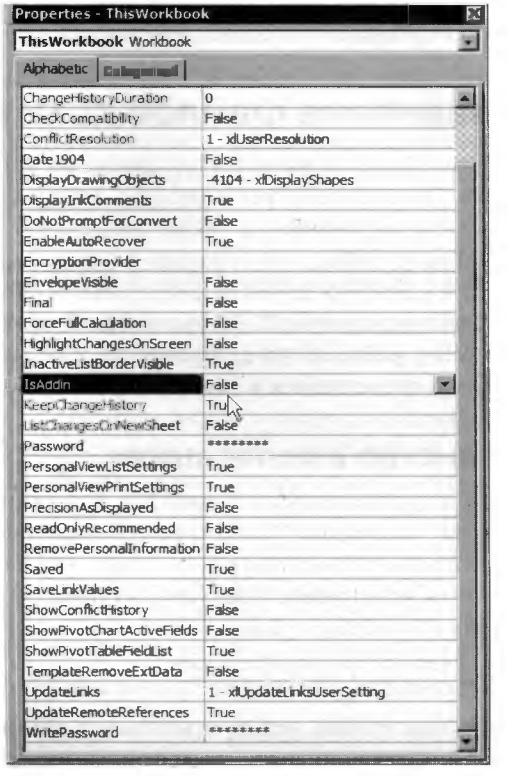


Рис. 21.5. Превращаем надстройку в обычную рабочую книгу

Действия, которые необходимо выполнить для создания надстройки

Перед распространением надстройки проверьте соответствие выполненных действий следующему списку.

- Проверена ли надстройка на всех поддерживаемых платформах и версиях Excel?
- Присвоено ли проекту VBA новое имя? По умолчанию каждый проект получает имя VBProject. Рекомендуется присвоить проекту описательное имя.
- Анализирует ли надстройка структуру и названия папок компьютера?
- Корректно ли отображаются имя и описание надстройки в диалоговом окне Надстройки (Add-Ins)?
- Если надстройка включает функции VBA, которые не предназначены для использования в рабочей книге, были ли эти функции объявлены с областью действия `Private`?
- Если не были, то эти функции отображаются в диалоговом окне Мастер функций (Insert Function).

- Не забыли ли вы удалить все операторы Debug⇨Print (Отладка⇨Печать) из кода?
- Выполнялась ли принудительная повторная компиляция надстройки, позволяющая удостовериться в отсутствии синтаксических ошибок?
- Учитываются ли вопросы интернационализации?
- Оптимизирована ли надстройка на оптимальное быстродействие? См. раздел "Оптимизация производительности надстроек".

Сравнение файлов XLAM и XLSM

Этот раздел начинается со сравнения файла надстройки XLAM с исходным файлом XLSM. Далее рассмотрены методы, которые применяются для оптимизации производительности надстройки, а также предложена методика, позволяющая уменьшить размер файла надстройки, что приведет к ускорению ее загрузки, а также к экономии места на диске и к освобождению дополнительной оперативной памяти.

Файл надстройки XLSM имеет такой же размер, как и исходный файл рабочей книги. Код VBA файла XLAM не сжимается и не оптимизируется, поэтому увеличение быстродействия не относится к достоинствам надстроек.

Членство в коллекциях

Надстройка является элементом коллекции AddIns, но не выступает "официальным членом" коллекции Workbooks. К надстройке можно обращаться с помощью метода Workbooks объекта Application, указав в качестве индекса имя файла, в котором хранится надстройка. Следующий оператор создает объект, который представляет надстройку Myaddin.xlam.

```
Dim TestAddin As Workbook
Set TestAddin = Workbooks("myaddin.xlam")
```

На надстройки нельзя ссылаться с помощью числового индекса коллекции Workbooks. Если применить приведенный далее код для просмотра членов этой коллекции, то надстройка Myaddin.xlam отображена не будет.

```
Dim w as Workbook
For Each w in Application.Workbooks
   MsgBox w.Name
Next w
```

С другой стороны, следующий цикл For Next позволяет отобразить надстройку Myaddin.xlam (если, конечно, Excel о ней знает) в диалоговом окне Надстройки (Add-Ins).

```
Dim a as Addin
For Each a in Application.AddIns
   MsgBox a.Name
Next a
```

Отображение окон файлов XSLM и XLAM

Обычные рабочие книги отображаются в одном или нескольких окнах. Например, представленный далее оператор отображает количество окон активной рабочей книги.

```
MsgBox ActiveWorkbook.Windows.Count
```

Состоянием отображения каждого окна рабочей книги можно управлять с помощью команды Вид⇨Окно⇨Скрыть (View⇨Window⇨Hide) или значений свойства `Visible`. Следующий код скрывает все окна активной рабочей книги.

```
Dim Win As Window
For Each Win In ActiveWorkbook.Windows
   Win.Visible = False
Next Win
```

Файлы надстроек никогда не отображаются и официально не имеют окон, хотя содержат рабочие листы (скрытые). Следовательно, в списке открытых окон меню Вид⇨Окно⇨Перейти в другое окно (View⇨Window⇨Switch Windows) надстройки не представлены. Если надстройка `Myaddin.xlam` открыта, то следующий оператор возвратит значение `0`.

```
MsgBox Workbooks("myaddin.xlam").Windows.Count
```

Рабочие листы и листы диаграмм в файлах XLSM и XLAM

Файлы надстроек, как и файлы рабочих книг, могут содержать любое количество рабочих листов или листов диаграмм. Но, как было отмечено ранее, файл XLSM должен иметь как минимум один рабочий лист, чтобы можно было преобразовать его в надстройку.

После открытия надстройки код VBA будет получать доступ к листам, содержащимся в надстройке, как и в случае обычной рабочей книги. В связи с тем, что файлы надстроек не являются частью коллекции `Workbooks`, на надстройку необходимо ссылаться по имени, а не по индексу. Из следующего примера вы узнаете, как извлекать значения ячейки A1 на первом рабочем листе надстройки `Myaddin.xlam`, открытой в момент выполнения этого кода.

```
MsgBox Workbooks("myaddin.xlam").Worksheets(1).Range("A1").Value
```

Если надстройка содержит рабочий лист, который должен отображаться, то можно скопировать его в открытую рабочую книгу или создать новую рабочую книгу на основе этого листа.

Например, представленный ниже код копирует первый рабочий лист надстройки и размещает его в активной рабочей книге (в качестве последнего листа рабочей книги).

```
Sub CopySheetFromAddin()
   Dim AddinSheet As Worksheet
   Dim NumSheets As Long
   Set AddinSheet = Workbooks("myaddin.xlam").Sheets(1)
   NumSheets = ActiveWorkbook.Sheets.Count
   AddinSheet.Copy After:=ActiveWorkbook.Sheets(NumSheets)
End Sub
```

Обратите внимание на то, что эта процедура работает даже в том случае, если VBA-проект надстройки защищен паролем.

Создать рабочую книгу на основе листа из надстройки еще проще.

```
Sub CreateNewWorkbook()
   Workbooks("myaddin.xlam").Sheets(1).Copy
End Sub
```

Примечание

В предыдущем примере предполагается, что код находится в файле, отличном от файла надстройки. Код VBA в файле надстройки всегда может использовать объект `ThisWorkbook` (ЭтаКнига) для определения ссылок на листы или диапазоны надстройки. Например, следующий оператор должен

находиться в модуле кода VBA надстройки. Приведенный ниже оператор отображает значение ячейки A1 листа Лист1.

```
MsgBox ThisWorkbook.Sheets("Лист1").Range("A1").Value
```

Получение доступа к VBA-процедурам надстройки

Доступ к процедурам VBA, которые сохраняются в надстройке, несколько отличается от доступа к процедурам файла обычной рабочей книги XLSM. Если выполнить команду Вид⇨Макросы⇨Макросы (View⇨Macros⇨Macros), то в диалоговом окне Макрос (Macro) не будет макросов, сохраненных в открытой надстройке. Это вызвано попыткой Excel ограничить доступ к таким процедурам.

Совет

Если имя процедуры известно, то его можно ввести непосредственно в диалоговом окне Макрос (Macro) и щелкнуть на кнопке Выполнить (Run) для запуска на выполнение. При этом процедура Sub должна находиться в модуле кода общего назначения, а не в модулях кода конкретных объектов.

Поскольку процедуры, находящиеся в надстройке, не отображаются в диалоговом окне Макрос (Macro), для доступа к ним используются другие средства. К возможным вариантам решения проблемы относятся использование стандартных методов (назначение комбинаций клавиш, команд ленты и элементов контекстного меню), а также косвенных методов (создание процедур обработки событий). Одним из главных выступает метод OnTime, который выполняет процедуру в определенное время суток.

Можно также воспользоваться методом Run объекта Application для выполнения процедуры, содержащейся в надстройке.

```
Application.Run "myaddin.xlam!DisplayNames"
```

Еще одним вариантом можно считать использование команды Tools⇨References (Сервис⇨Ссылки) редактора VBE, которая позволяет создать ссылку на надстройку. В итоге вы непосредственно ссылаетесь на одну из процедур в коде VBA, не указывая имени файла. На самом деле необходимости в использовании метода Run нет: процедуру можно вызывать непосредственно, пока она не объявлена с ключевым словом Private. Представленный далее оператор выполняет процедуру DisplayNames, которая содержится в надстройке, указанной с помощью команды Tools⇨References.

```
Call DisplayNames
```

Примечание

Даже после установки ссылок на надстройку имена содержащихся в ней макросов не отображаются в диалоговом окне Макрос (Macro).

Функции, определенные в надстройке, работают точно так же, как и функции, определенные в рабочей книге XLSM. Доступ к подобным функциям не представляет особого труда, поскольку Excel отображает их имена в диалоговом окне Вставка функции (Insert Function), в категории Определенные пользователем (User Defined), по умолчанию. Исключение состоит в том, что, если функция определяется с помощью ключевого слова Private, она не отображается в этом списке. Объявляйте функцию с помощью ключевого слова Private, если она будет использоваться только другими процедурами VBA, а не формулами рабочего листа.

Примечание

Примером надстройки, функции которой *не* объявляются с помощью ключевого слова `Private`, является мастер подстановок (Lookup Wizard). Эта надстройка входила в комплект поставки более ранних версий Excel, сейчас же ее можно загрузить с веб-сайта Microsoft. Установите ее и щелкните на кнопке Вставить функцию (Insert Function). В категории Определенные пользователем (User Defined) диалогового окна Вставка функции (Insert Function) находится более трех десятков функций, не являющихся функциями рабочего листа (рис. 21.6). Эти функции не предназначены для применения в формулах рабочего листа, но будут отображены в списке, если при их объявлении не используется ключевое слово `Private`.

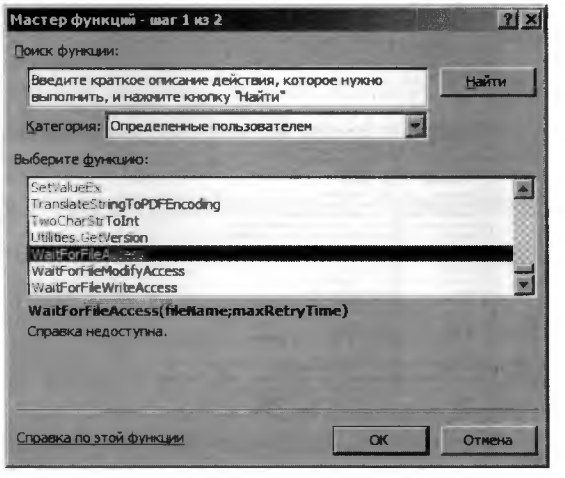


Рис. 21.6. Если при объявлении этих функций используется ключевое слово `Private`, они не будут отображаться в этом окне

Как отмечалось ранее, можно использовать функции рабочего листа, находящиеся в надстройке, без указания имени рабочей книги. Например, если пользовательская функция MOVAVG хранится в файле `newfuncs.xlsm`, воспользуйтесь следующей инструкцией для обращения к функции из рабочего листа, который находится в другой книге:

```
=newfuncs.xlsm!MOVAVG(A1:A50)
```

Но если эта функция хранится в открываемом файле надстройки, можно не указывать ссылку на файл, записав следующую формулу:

```
=MOVAVG(A1:A50)
```

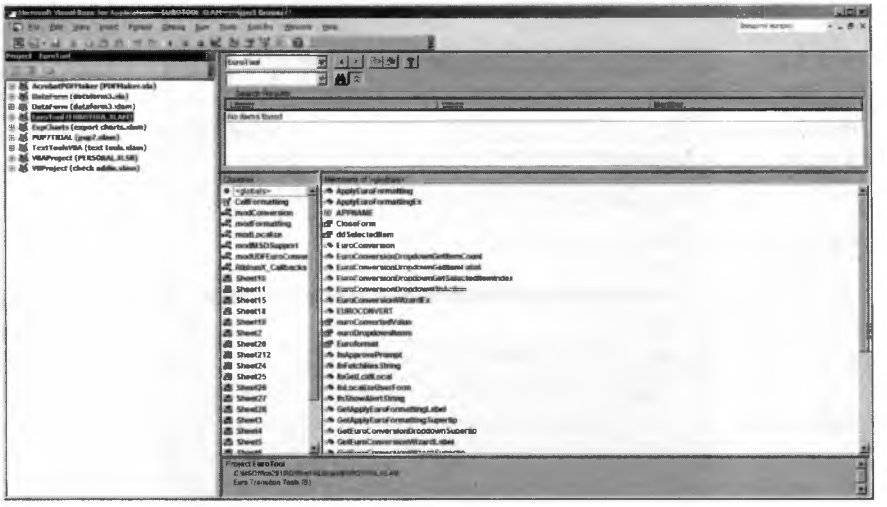
Поиск в защищенной надстройке

Диалоговое окно Макрос (Macro) не отображает имена процедур, которые содержатся в надстройках. Однако иногда нужно запускать такую процедуру, когда надстройка защищена (это не позволит обратиться к исходному коду для определения имени процедуры). Для решения подобной проблемы воспользуйтесь окном Object Browser.

Для иллюстрации сказанного установите надстройку Инструменты для евро (Euro Currency Tools). Она распространяется вместе с Excel и защищена от просмотра, что не позволяет просматривать исходный код ее макросов. После установки этой надстройки соз-

дается новая группа Solutions (Решения), находящаяся на вкладке ленты Формулы (Formulas). После щелчка на кнопке Euro Conversion (Конвертирование евро) отображается диалоговое окно Euro Conversion (Конвертирование евро). В нем можно конвертировать валюты.

1. Активизируйте редактор VBE и выберите проект `EUROTOOL.XLAM` в окне Project (Проект).
2. Нажмите клавишу <F2> для активизации окна Object Browser.
3. В раскрывающемся списке Libraries (Библиотеки) выберите пункт `EuroTool`. Это приведет к отображению всех классов надстройки `EUROTOOL.XLAM`, как показано на следующем рисунке.



4. Выберите различные элементы в списке Classes (Классы), чтобы ознакомиться с классами и их элементами.

Эта надстройка включает 3 окна `UserForm`, 6 модулей VBA и 19 листов. В Excel можно копировать листы из защищенных настроек, благодаря чему появляется возможность их просмотра. Для копирования листа надстройки в новую рабочую книгу в окне Immediate введите следующий оператор.

```
Workbooks("eurotool.xlam").Sheets(1).Copy
```

Если же нужно просмотреть все листы надстройки, воспользуйтесь следующим оператором.

```
Workbooks("eurotool.xlam").IsAddin = False
```

На приведенном ниже рисунке представлена часть листа, скопированного из надстройки `EUROTOOL.XLAM`. Этот лист (и другие листы) содержит информацию, которая используется для локализации надстройки на других языках.

Все, что перечислено выше, интересно, но вряд ли поможет идентифицировать имя процедуры.

Эта надстройка включает множество процедур. Я попытался выбрать из них те, которые отображают диалоговые окна с требуемыми именами. Первая попытка завершилась неудачей. Затем я обратился к объектам, перечисленным в модуле кода `ThisWorkbook`, где увидел подходящую процедуру `EuroConversionWizard`. После ее запуска появилось сообщение об ошибке. Затем я попытался выполнить следующую команду:

```
Application.Run "eurotool.xlam!ThisWorkbook.EuroConversionWizard"
```

	A	B
1	frmEuro	
2	Accelerator	Caption
3		유로 변환
4		유로로 변환할 데이터
5	S	원본 범위(S):
6	D	대상 범위(D):
7		통화 변환
8	F	원본(F):
9	T	대상(T):
10	O	산출 형식(O):
11	A	고급(A)
12		확인
13		취소
14		<unused>

Наконец-то удача! В результате выполнения указанной выше команды на экране появилось диалоговое окно Euro Conversion.

Вооружившись этой информацией, можно написать код VBA для отображения диалогового окна Euro Conversion (при необходимости в этом).

Управление надстройками с помощью кода VBA

Информация в этом разделе поможет вам создавать процедуры VBA, используемые для управления надстройками.

Коллекция `AddIns` состоит из объектов всех надстроек, о которых "знает" Excel. В нее входят как установленные, так и неустановленные надстройки. В диалоговом окне Надстройки (Add-Ins) перечислены все элементы коллекции `AddIns`. Возле названий надстроек расположены флажки, выставленные для установленных надстроек.

Новинка

В Excel 2010 появилась дополнительная коллекция под названием `AddIns2`. Она совпадает с коллекцией `AddIns` и дополнительно включает надстройки, которые можно открыть с помощью команды File⇨Open (Файл⇨Открыть). Ранее для доступа к подобным надстройкам требовались макросы XLM.

Добавление элемента в коллекцию `AddIns`

Файлы надстроек, составляющие коллекцию `AddIns`, могут находиться в любой папке. Excel сохраняет неполный список файлов и папок, в которых они находятся, в системном реестре Windows. Excel 2010 хранит эту информацию в следующей ветви системного реестра:

```
HKEY_CURRENT_USER\Software\Microsoft\Office\14.0\Excel\Add-in Manager
```

Можно воспользоваться редактором реестра Windows (Regedit.exe) для просмотра соответствующих параметров реестра. Обратите внимание на то, что "стандартные" надстройки, которые поставляются вместе с Excel, не указаны в этой записи реестра. Кроме того, файлы надстроек, которые хранятся в указанной ниже папке, также будут добавлены в список (хотя и не будут указаны в системном реестре).

```
C:\Program Files\Microsoft Office\Office14\Library
```

Можете добавить новый объект `AddIn` в коллекцию `AddIns` как вручную, так и программно с помощью VBA. Для того чтобы вручную добавить новую надстройку в кол-

лекцию, перейдите в диалоговое окно Надстройки (Add-Ins), щелкните на кнопке Просмотр (Browse) и найдите файл необходимой надстройки.

Добавить новый элемент в коллекцию `AddIns` можно с помощью метода `Add` коллекции `AddIns`.

```
Application.AddIns.Add "c:\files\newaddin.xlam"
```

После выполнения представленной выше операции в коллекции `AddIns` будет содержаться дополнительный элемент, а в диалоговом окне Надстройки (Add-Ins) отобразится новый элемент списка. Если надстройка уже существует в коллекции, то ничего не произойдет, и сообщение об ошибке не будет генерироваться.

Когда добавляемая надстройка находится на сменном носителе (например, на компакт-диске), можно скопировать файл в папку библиотеки Excel с помощью метода Add. В следующем примере файл `Myaddin.xlam` копируется с диска E и добавляется в коллекцию `AddIns`. Второй аргумент (в данном случае равный `True`) указывает на необходимость копирования надстройки. Если надстройка находится на жестком диске, то второй аргумент можно игнорировать.

```
Application.AddIns.Add "e:\myaddin.xla", True
```

Примечание

Добавление новой надстройки в коллекцию `AddIns` не приводит к ее установке. Для установки надстройки присвойте ее свойству `Installed` значение `True`.

Предупреждение

Системный реестр Windows не обновляется до тех пор, пока Excel не завершит свою работу. Если Excel завершает работу нестандартным образом (в случае системного сбоя), имя надстройки не добавляется в системный реестр, а сама надстройка не становится частью коллекции `AddIns` при следующем запуске Excel.

Удаление элемента из коллекции `AddIns`

Как это ни удивительно, однако не существует явного способа удаления элемента из коллекции `AddIns`. В коллекции `AddIns` нет метода `Delete` или `Remove`. Одним из способов удаления надстройки из диалогового окна Надстройки (Add-Ins) является непосредственное редактирование параметра системного реестра (для этого используется программа regedit.exe). В результате надстройка не будет отображена в списке диалогового окна Надстройки при следующем запуске Excel. Обратите внимание на то, что данный метод не применим ко всем файлам надстроек.

Еще одним способом удаления надстройки из коллекции `AddIns` является удаление, перемещение или переименование файла XLAM (или XLA), который содержит надстройку. При последующей попытке установки этой надстройки будет отображено предупреждение, которое показано на рис. 21.7. В этом предупреждении пользователю предоставляется возможность удалить надстройку из коллекции `AddIns`.

Рис. 21.7. Весьма хитроумный способ удаления элемента коллекции `AddIns`

войства объекта AddIn

Объект AddIn является единственным представителем коллекции AddIns. Например, я того чтобы отобразить имя файла, в котором содержится надстройка, являющаяся пер-м членом коллекции AddIns, можно воспользоваться следующим оператором:

```
gbox AddIns(1).Name
```

Объект AddIn имеет 14 свойств, информацию о которых можно получить в справоч-м руководстве. Пять свойств являются "скрытыми". Их названия довольно запутаны, этому мы рассмотрим некоторые наиболее важные свойства этого объекта.

ойство Name

Это свойство содержит имя файла надстройки. Name — свойство, предназначенное лько для чтения; оно не позволяет изменить имя файла и указать другое значение ойства Name.

ойство Path

Свойство, содержащее имя диска и папки, в которых расположен файл надстройки. ачение этого свойства не имеет в конце наклонной черты и имени файла.

ойство Fullname

Это свойство содержит полное имя надстройки, которое состоит из имени диска, пути имени файла. Данное свойство бесполезно, поскольку такая же информация предос-вляется свойствами Name и Path. Следующие операторы в результате выполнения звращают одинаковые строки.

```
sgBox AddIns(1).Path & "\" & AddIns(1).Name
sgBox AddIns(1).FullName
```

ойство Title

Скрытое свойство, содержащее описательное имя надстройки. Свойство Title ото-ажается в диалоговом окне Надстройки (Add-Ins). Данное свойство предназначено лько для чтения, и единственный способ изменить его значение — использовать па-ель Свойства документа (Document Properties). Для отображения этой панели вос-льзуйтесь командой Разработчик⇨Изменение⇨Область документа (Developer⇨ lodify⇨Document). Эту команду следует применить к XLSM-версии файла (до его пре-бразования в надстройку). Еще один способ отобразить эту панель — щелкнуть правой нопкой мыши на файле надстройки в окне проводника Windows и в контекстном меню ыбрать параметр Свойства (Properties). Затем нужно выбрать вкладку Подробно Details) и выполнить необходимые изменения. Этот метод не работает в случае, если айл открыт в Excel.

Как правило, к члену коллекции обращаются с помощью значения свойства Name. оллекция AddIns этим отличается от других коллекций. В ней вместо свойства Name ля адресации используется значение свойства Title. В следующем примере отобража-тся имя файла надстройки Analysis ToolPak (Пакет анализа), которая находится в файле nalys32.xll, а свойству Title присвоено значение "Analysis ToolPak".

```
ub ShowName()
  MsgBox AddIns("Analysis Toolpak").Name
nd Sub
```

Можно, конечно, ссылаться на определенную надстройку по индексу, если индекс известен точно. Но в большинстве случаев для ссылки на надстройку используется ее свойство Name.

Свойство Comments

В этом скрытом свойстве хранится текст, который отображается в диалоговом окне Надстройки (Add-Ins), описывающем определенную надстройку. Свойство Comments предназначено только для чтения. Единственный способ изменить его значение — использовать диалоговое окно Свойства (Properties) до преобразования рабочей книги в надстройку. Описание может состоять из 255 символов, но диалоговое окно Надстройки отображает из них только первые 100.

Если макрос попытается считать значение свойства Comments надстройки, не содержащей комментариев, отобразится сообщение об ошибке.

Свойство Installed

Свойство Installed имеет значение True, если надстройка установлена (т.е. в диалоговом окне Надстройки (Add-Ins) напротив имени надстройки установлен флажок). Установка свойства Installed равной значению True приводит к загрузке надстройки. Установка этого свойства равной значению False приводит к выгрузке надстройки из памяти. Ниже приведен пример установки (т.е. открытия) надстройки Analysis ToolPak (Пакет анализа) с помощью кода VBA.

```
Sub InstallATP()
   AddIns("Analysis ToolPak").Installed = True
End Sub
```

После выполнения процедуры в диалоговом окне Надстройки (Add-Ins) будет отображаться установленный флажок напротив имени надстройки Analysis ToolPak (Пакет анализа). Если надстройка уже установлена, то присвоение свойству Installed значения True не приведет к выполнению каких-либо действий. Для того чтобы отключить надстройку (удалить), установите свойство Installed равным False.

Предупреждение

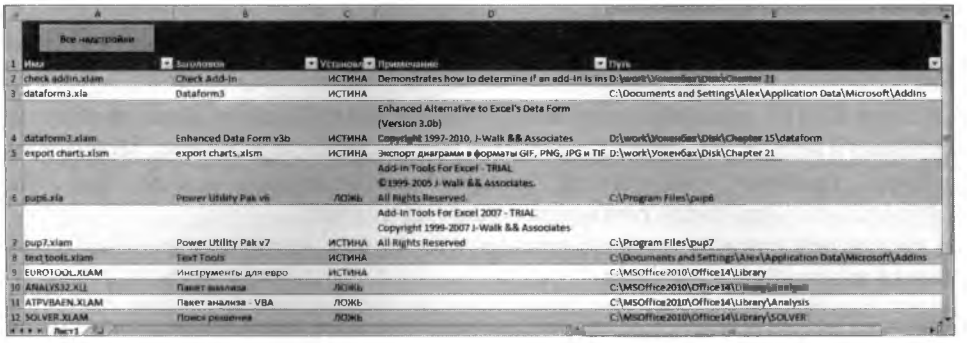
Если файл надстройки был открыт с помощью команды Файл⇨Открыть (File⇨Open), он не рассматривается как "официально" установленный. Следовательно, свойству Installed будет присвоено значение False.

Процедура ListAllAddIns, код которой приводится ниже, отображает все элементы коллекции AddIns, а также свойства Name, Title, Installed, Comments и Path.

```
Sub ListAllAddins()
   Dim ai As AddIn
   Dim Row As Long
   Dim Table1 As ListObject
   Cells.Clear
   Range("A1:E1") = Array("Name", "Title", "Installed", _
      "Comments", "Path")
   Row = 2
   On Error Resume Next
   For Each ai In AddIns
      Cells(Row, 1) = ai.Name
      Cells(Row, 2) = ai.Title
      Cells(Row, 3) = ai.Installed
```

```
        Cells(Row, 4) = ai.Comments
        Cells(Row, 5) = ai.Path
        Row = Row + 1
    Next ai
    On Error GoTo 0
    Range("A1").Select
    ActiveSheet.ListObjects.Add
    ActiveSheet.ListObjects(1).TableStyle = _
        "TableStyleMedium2"
End Sub
```

На рис. 21.8 показан результат выполнения этой процедуры.



	A	B	C	D	E
1	Все надстройки				
2	Имя	Заголовок	Установлен	Примечание	Путь
3	check addin.xlam	Check Add-In	ИСТИНА	Demonstrates how to determine if an add-in is ins	D:\work\Уокенбах\Disk\Chapter 21
4	dataform3.xla	Dataform3	ИСТИНА		C:\Documents and Settings\Alex\Application Data\Microsoft\AddIns
5	dataform2.xlam	Enhanced Data Form v3b	ИСТИНА	Enhanced Alternative to Excel's Data Form (Version 3.0b) Copyright 1997-2010, J-Walk && Associates	D:\work\Уокенбах\Disk\Chapter 15\dataform
6	export charts.xlsm	export charts.xlsm	ИСТИНА	Экспорт диаграмм в форматы GIF, PNG, JPG и TIF	D:\work\Уокенбах\Disk\Chapter 21
7	pup6.xla	Power Utility Pak v6	ЛОЖЬ	Add-In Tools For Excel - TRIAL ©1995-2005 J-Walk && Associates. All Rights Reserved.	C:\Program Files\pup6
8	pup7.xlam	Power Utility Pak v7	ИСТИНА	Add-In Tools For Excel 2007 - TRIAL Copyright 1999-2007 J-Walk && Associates All Rights Reserved	C:\Program Files\pup7
9	text tools.xlam	Text Tools	ИСТИНА		C:\Documents and Settings\Alex\Application Data\Microsoft\AddIns
10	EUROTOOL.XLAM	Инструменты для евро	ИСТИНА		C:\MSOffice2010\Office14\Library
11	ANALYS32.XLL	Пакет анализа	ЛОЖЬ		C:\MSOffice2010\Office14\
12	ATPVBAEN.XLAM	Пакет анализа - VBA	ЛОЖЬ		C:\MSOffice2010\Office14\Library\Analysis
13	SOLVER.XLAM	Поиск решения	ЛОЖЬ		C:\MSOffice2010\Office14\Library\SOLVER

Рис. 21.8. Элементы коллекции `AddIns`

Компакт-диск

Эта процедура находится на прилагаемом к книге компакт-диске в файле `list add-in information.xlsm`.

Примечание

Для определения того, является ли рабочая книга надстройкой, проверьте значение свойства `IsAddIn`. Это свойство может быть изменено пользователем, поэтому преобразовать рабочую книгу в надстройку можно, присвоив свойству `IsAddIn` значение `True`.

Можно также преобразовать надстройку в рабочую книгу, присвоив свойству `IsAddIn` значение `False`. После этого листы надстройки будут отображаться в окне Excel, даже если проект VBA надстройки является защищенным.

Получение доступа к надстройке как к рабочей книге

Как уже отмечалось, существует два метода открытия надстройки: с помощью диалогового окна Надстройки (Add-Ins) и с помощью команды Файл⇨Открыть (File⇨Open). Первый метод более предпочтителен при управлении надстройками. На то существует особая причина: при открытии надстройки с помощью команды Файл⇨Открыть свойство `Installed` не устанавливается равным значению `True`. Таким образом, данный файл невозможно закрыть с помощью диалогового окна Надстройки.

Единственной возможностью закрыть эту надстройку является использование соответствующего оператора VBA. Например, можно применить следующий оператор:

```
Workbooks("myaddin.xlam").Close
```

Предупреждение

Использование метода `Close` по отношению к установленной надстройке приводит к ее удалению из памяти, но не присваивает свойству `Installed` значения `False`. В результате в диалоговом окне Надстройки (Add-Ins) продолжают отображаться установленные надстройки. Правильный способ удаления установленных надстроек — присвоить свойству `Installed` значение `False`.

Итак, как вы заметили, возможность использования надстроек Excel несколько неоднозначна. Разработчик должен быть особо внимателен к вопросам установки и удаления надстроек.

События объекта `AddIn`

Объекту `AddIn` присущи два события: `AddInInstall` (возникает при установке надстройки) и `AddInUninstall` (возникает при удалении надстройки). Можно создать процедуры обработки этих событий, которые размещаются в модуле кода объекта `ThisWorkbook` (ЭтаКнига), соответствующего надстройке.

В следующем примере отображается окно сообщения, которое появляется в момент установки надстройки.

```
Private Sub Workbook_AddInInstall()
MsgBox ThisWorkbook.Name & " надстройка деинсталлирована."
End Sub
```

Предупреждение

Не путайте событие `AddInInstall` с событием `Open`. Событие `AddInInstall` происходит в том случае, когда надстройка устанавливается впервые, а не при каждом ее открытии. Если нужно вызывать код при каждом открытии надстройки, используйте процедуру `Workbook_Open`.

Перекрестная ссылка

Дополнительные сведения о событиях можно найти в главе 19.

Оптимизация производительности надстроек

Если попросить десять программистов провести автоматизацию определенной задачи, то вполне вероятно, что в результате будет получено десять различных решений. Чаще всего производительность всех этих решений будет находиться на одинаковом уровне.

Приведенные ниже рекомендации можно использовать для обеспечения максимального быстродействия кода — они позволят ускорить выполнение любого кода VBA, а не только кода надстроек.

- **Установите свойство `Application.ScreenUpdating` равным значению `False`**, если выполняется запись данных на рабочий лист выполняются другие действия, приводящие к изменению отображаемых данных.
- **Объявите тип данных всех применяемых переменных, а в завершенном коде любой ценой избегайте использования переменных типов.** Используйте опера-

тор `Option Explicit` в начале каждого модуля, чтобы сделать объявление типа переменной обязательным.

- **Создайте объектные переменные, чтобы избежать длинных ссылок на объекты.** Например, если вы управляете объектом `Series` диаграммы, то создайте переменную с помощью следующего кода.

```
Dim S1 As Series
Set S1 = ActiveWorkbook.Sheets(1).ChartObjects(1). _
   Chart.SeriesCollection(1)
```

- **Объявите объектные переменные конкретного типа.** Избегайте использования объявления `As Object`.
- **Примените конструкцию `With-End With`** для вызова нескольких методов и для изменения нескольких свойств одного объекта везде, где это возможно.
- **Удалите избыточный код**, особенно в записанных макросах.
- **Если возможно, управляйте данными с помощью массивов VBA, а не диапазонов рабочих листов.** Чтение и запись на рабочий лист отнимают намного больше времени, чем обработка данных в памяти. Но это правило не всегда справедливо. Для получения наилучшего результата необходимо проверить быстродействие обоих вариантов.
- **Если код записывает большое количество данных в рабочие листы, выберите ручной режим вычислений.** В результате существенно возрастает скорость вычислений. Для изменения режима вычислений можно воспользоваться следующим оператором:

```
Applicatiuon.Calculation = xlCalculationManual
```

- **Избегайте связывания элементов управления пользовательских диалоговых окон с ячейками на рабочем листе**, что может привести к пересчету рабочего листа при каждом изменении элемента управления в диалоговом окне.
- **Перед созданием надстройки скомпилируйте используемый код.** Это приведет к увеличению размера файла, но избавит от необходимости компилировать код перед каждым запуском процедур.

Проблемы, связанные с использованием надстроек

Надстройки являются достаточно мощным средством. Однако при их создании и использовании вас подстерегают некоторые сложности. Надстройки "страдают" собственным набором проблем, часть которых можно решить. В этом разделе будут рассмотрены вопросы, на которые необходимо обратить особое внимание при создании надстроек для массового распространения среди пользователей.

Правильная установка

В некоторых случаях нужно правильно установить надстройку, которая была открыта с помощью диалогового окна Надстройки (Add-Ins), а не команды Файл⇨Открыть (File⇨Open). В этом разделе описывается методика, которая позволяет определить, открыта ли данная надстройка, а также предоставляет пользователю возможность установить надстройку, если она установлена неправильно.

Если надстройка установлена некорректно, отображается соответствующее сообщение (рис. 21.9). Щелкните на кнопке Да (Yes) для установки надстройки. Щелчок на кнопке Нет (No) приводит к тому, что файл надстройки остается открытым, но надстройка не устанавливается.

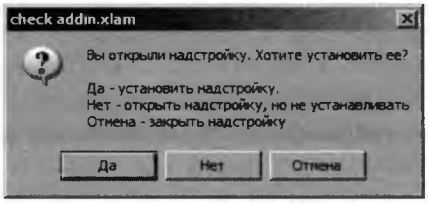


Рис. 21.9. Это сообщение отображается при открытии некорректно установленной надстройки

Чтобы закрыть файл, щелкните на кнопке Отмена (Cancel). Ниже представлен код, который находится в модуле кода объекта ThisWorkbook (ЭтаКнига). Эта методика основана на том факте, что событие AddInInstall происходит до события Open рабочей книги.

```
Dim InstalledProperly As Boolean
Private Sub Workbook_AddinInstall()
   InstalledProperly = True
End Sub
Private Sub Workbook_Open()
   Dim ai As AddIn, NewAi As AddIn
   Dim M As String
   Dim Ans As Integer
'  Была ли надстройка установлена с помощью окна Надстройки?
   If InstalledProperly Then Exit Sub

'  Находится ли объект в коллекции AddIns?
   For Each ai In AddIns
      If ai.Name = ThisWorkbook.Name Then
         If ai.Installed Then
            MsgBox "Надстройка установлена корректно.", _
               vbInformation, ThisWorkbook.Name
            Exit Sub
         End If
      End If
   Next ai

'  Если не элемент коллекции AddIns, запрос пользователю.
   M = "Вы открыли надстройку. Хотите ее установить?"
   M = M & vbNewLine
   M = M & vbNewLine & "Да - установить надстройку. "
   M = M & vbNewLine & "Нет - открыть надстройку, _
      но не устанавливать."
   M = M & vbNewLine & "Отмена - закрыть надстройку"
   Ans = MsgBox(M, vbQuestion + vbYesNoCancel, _
      ThisWorkbook.Name)
   Select Case Ans
      Case vbYes
         ' Добавление в коллекцию AddIns и установка.
         Set NewAi = _
```

```
            Application.AddIns.Add(ThisWorkbook.FullName)
         NewAi.Installed = True
      Case vbNo
         ' Действие не выполняется, остается открытым
      Case vbCancel
         ThisWorkbook.Close
   End Select
End Sub
```

Эта процедура имеет следующие особенности.

- Надстройка может открываться автоматически, если она установлена и отображается в диалоговом окне Надстройки (Add-Ins). Сообщение на экране не отображается.
- Пользователь может установить надстройку в диалоговом окне Надстройки (Addins). Сообщение не отображается.
- Надстройку можно открыть вручную (с помощью команды Файл⇨Открыть (File⇨Open). Она не входит в коллекцию `AddIns`. При этом на экране отображается сообщение, в ответ на которое пользователь может выполнить одно из трех действий.
- Надстройку можно открыть вручную (с помощью команды Файл⇨Открыть (File⇨Open). Она входит в коллекцию `AddIns`. При этом на экране отображается сообщение, в ответ на которое пользователь может выполнить одно из трех действий.

Этот код может также использоваться для упрощения установки надстройки, которую вы кому-либо передаете в дальнейшем. Для установки этой надстройки достаточно дважды щелкнуть на ее имени (при этом она открывается в Excel) и ответить "Да" в ответ на запрос. А еще лучше — модифицировать код таким образом, чтобы надстройка устанавливалась без запроса.

Компакт-диск

Эта надстройка находится на прилагаемом к книге компакт-диске в файле `check addin.xlam`. Попытайтесь открыть ее с помощью одного из двух методов (в диалоговом окне Надстройки (Add-Ins) или с помощью команды Файл⇨Открыть (File⇨Open)).

Ссылки на другие файлы

Если надстройка использует другие файлы, то необходимо быть особенно внимательным при распространении приложения. Нельзя ничего предположить о файловой структуре системы, в которой пользователь запускает приложение. Самый простой метод — потребовать от пользователя расположить все файлы приложения в одной папке. Затем используйте свойство `Path` рабочей книги приложения, чтобы указать ссылки на необходимые файлы.

Например, если приложение имеет собственную справочную систему, то обязательно удостоверьтесь, что этот файл находится в той же папке, что и само приложение. Затем можете использовать представленную ниже процедуру для поиска файла справочных сведений.

```
Sub GetHelp()
   Application.Help ThisWorkbook.Path & "\userhelp.chm "
End Sub
```

Если приложение использует API-функции стандартных библиотек Windows, то можно ожидать, что Windows найдет все необходимые файлы. Однако если используется собственная библиотека DLL, то лучшим выходом будет сохранение этой библиотеки в

папке Windows\System (которая может называться и по-другому). Для определения пути к папке System воспользуйтесь функцией Windows API GetSystemDirectory.

Указание правильной версии Excel

Пользователи более ранних версий Excel могут открывать файлы Excel 2007 (либо более поздней версии) с помощью утилиты Compatibility Pack, разработанной компанией Microsoft. Если созданная вами надстройка обладает свойствами, которые присущи только Excel 2007 либо Excel 2010, следует предупреждать об этом пользователей более ранних версий Excel. Ниже приводится используемый в этом случае код.

```
Sub CheckVersion()
   If Val(Application.Version) < 12 Then
      MsgBox "Этот код работает только в Excel 2007 или _
         в более поздних версиях"
      ThisWorkbook.Close
   End If
End Sub
```

Свойство Version объекта Application возвращает строку. Например, может быть возвращено значение 12.0a. Эта процедура использует функцию VBA Val, которая игнорирует все, за исключением первого символа алфавита.

Перекрестная ссылка

Обратитесь к главе 26 за дополнительными сведениями о совместимости.

Часть VI

Разработка приложений

В этой части...

Глава 22

Работа с лентой

В этой главе...

- Начальные сведения о ленте
- Управление лентой с помощью VBA
- Настройка ленты
- Создание "старомодных" панелей инструментов

Начиная с версии Excel 2007 лента формирует совершенно новый облик пользовательского интерфейса. Для изменения ленты используются возможности языка XML, хотя некоторые настройки могут быть выполнены с помощью VBA.

Начальные сведения о ленте

Начиная с версии Excel 2007 на смену испытанным временем меню и панелям инструментов пришел новый интерфейс, основанный на *ленте и вкладках*. Новый интерфейс радикальным образом отличается от прежнего, хотя его и можно рассматривать в качестве некоего симбиоза используемых ранее меню и панелей инструментов.

На протяжении многих лет пользователи Excel жаловались на то, что система меню этой замечательной программы усложняется с каждой новой версией. Кроме того, количество панелей инструментов превысило все разумные пределы. А еще приходится обеспечивать доступность каждого нового свойства этой программы. Ранее это обеспечивалось путем добавления новых элементов в меню и формирования новых панелей инструментов. Разработчики из компании Microsoft решили подобную проблему путем разработки нового интерфейса в форме ленты.

Реакция пользователей на появление ленты Office была неоднозначной. Лично мне лента пришлась по душе. После более чем трехлетней работы с Excel 2007 меня вовсе не прельщает идея возврата к устаревшей системе меню в Excel 2003.

Опытным пользователям, скорее всего, не понравится то, что многие привычные им последовательности команд не работают. Начинающим пользователям понравится то, что множество меню и панелей инструментов будут заменены простым и понятным интерфейсом в виде ленты, значительно ускоряющим работу.

В следующих разделах приводится достаточно подробное описание ленты, которое будет особенно полезным для начинающих пользователей.

Состав команд на ленте варьируется в зависимости от выбранной вкладки. На ленте находятся группы связанных команд. Ниже приводится краткий обзор вкладок Excel.

- **Файл (File).** Эта вкладка включает ряд команд, предназначенных для работы с файлами (копирование, удаление, печать и т.д.).
- **Главная (Home).** Именно на этой вкладке пользователь Excel проведет большую часть времени. Здесь находятся команды по работе с буфером обмена, команды форматирования и применения стилей, команды для вставки и удаления строк/столбцов, а также "джентльменский" набор команд по редактированию листов.
- **Вставка (Insert).** Эта вкладка используется для вставки различных объектов на рабочий лист — таблиц, диаграмм, символов и т.д.
- **Разметка страницы (Page Layout).** Эта вкладка включает команды, которые позволяют изменять внешний вид всего рабочего листа, а также изменять параметры печати.
- **Формулы (Formulas).** Применяйте эту вкладку для вставки формул, имен диапазонов, а также для доступа к средствам контроля вычислений, выполняемых Excel.
- **Данные (Data).** На этой вкладке находятся команды, связанные с обработкой данных Excel.
- **Рецензирование (Review).** Здесь находятся команды, с помощью которых выполняется проверка грамматики, перевод текста, добавление комментариев и установка защиты листов.
- **Вид (View).** На этой вкладке помещены команды, контролирующие различные аспекты просмотра листов. Некоторые команды этой вкладки также доступны в строке состояния.
- **Разработчик (Developer).** Эта вкладка по умолчанию не отображается. Здесь находятся команды, применяемые программистами. Для отображения этой вкладки выберите команду Файл⇨Параметры Excel (File⇨Excel Options) и перейдите в раздел Основные (Popular). Установите флажок Показывать вкладку "Разработчик" на ленте (Show Developer Tab in the Ribbon).
- **Надстройки (Add-Ins).** Эта вкладка отображается только в том случае, если была загружена рабочая книга или надстройка, которая настраивает меню или панели инструментов (с помощью объектов `CommandBar`). Поскольку панели инструментов и меню в Excel 2010 не поддерживаются, соответствующие настройки находятся на вкладке Надстройки.

Ширина ленты напрямую связана с величиной диалогового окна Excel. Если окно имеет минимальные размеры, большинство кнопок команд "исчезают" с ленты. На рис. 22.1 показаны три варианта вкладки ленты Главная (Home), имеющие различную ширину. На верхнем изображении отображены все элементы управления. На среднем изображении лента имеет среднюю ширину. Обратите внимание на то, что значки становятся меньше, а описательный текст исчезает. И самый экстремальный случай представлен на нижнем изображении, где лента имеет минимальную ширину. По причине отсутствия места в некоторых группах команд отображается всего лишь один значок. Но это не значит, что другие значки исчезают, — достаточно щелкнуть на значке мышью, как тут же отобразятся все команды из этой группы.

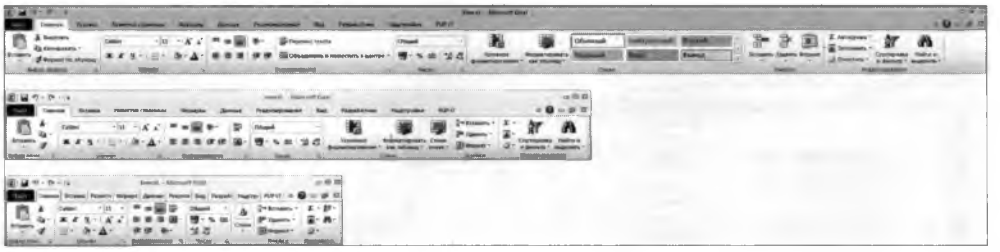


Рис. 22.1. Три различных представления вкладки ленты Главная

Совет

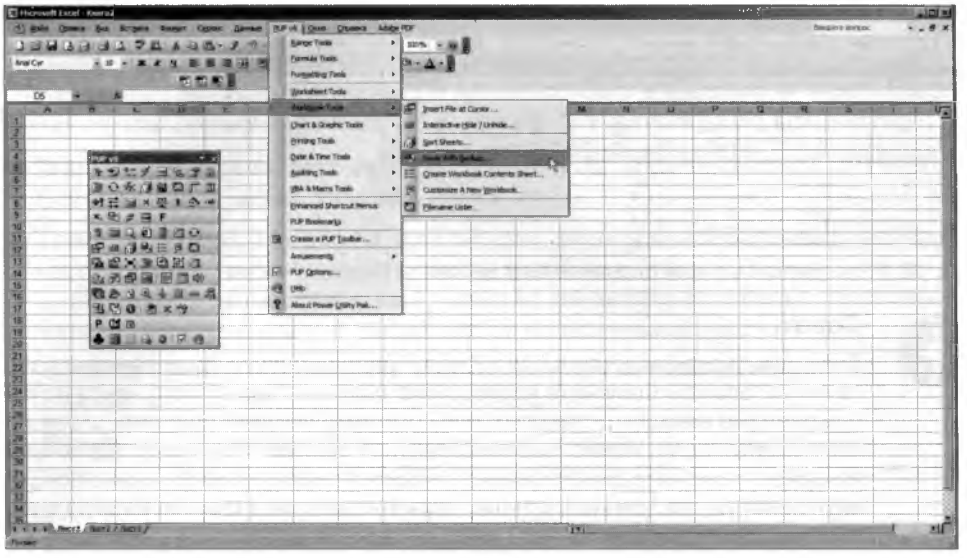
Если нужно скрыть ленту в целях увеличения рабочего пространства рабочего листа, дважды щелкните на любой из вкладок. После этого лента исчезает с экрана (но вкладки остаются), и вы сможете увидеть еще пять строк рабочего листа. Если нужно воспользоваться лентой повторно, щелкните на вкладке, после чего она временно появится снова. Если же нужно восстановить ленту на постоянной основе, дважды щелкните мышью на вкладке. Для отображения/сокрытия ленты можно также воспользоваться комбинацией клавиш <Ctrl+F1> либо щелкнуть на значке ^, расположенном справа от значка справки в строке заголовка Excel.

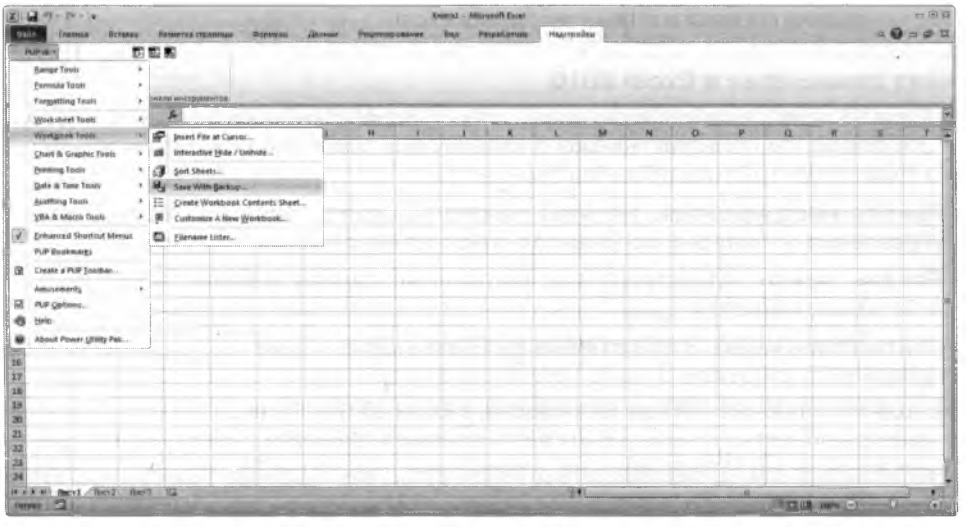
Объект `CommandBar` в Excel 2010

В Excel 97 появился абсолютно новый способ обработки меню и панелей инструментов. Эти элементы пользовательского интерфейса являются объектами `CommandBar`. Всего существует три типа объектов `CommandBar`.

- **Панель инструментов.** На этой панели находится один или более элементов управления, на которых могут выполняться щелчки мышью.
- **Панель меню.** Существуют две встроенные панели меню — `Лист` (Worksheet) и `Диаграмма` (Chart).
- **Контекстное меню.** Этот тип меню отображается на экране после щелчка на объекте правой кнопкой мыши.

Исходя из соображений совместимости, в Excel 2007 и Excel 2010 по-прежнему поддерживается объект `CommandBar`, хотя его функциональность существенно ограничена (пользователи не могут создавать настраиваемые панели инструментов). Но пользователи VBA, как и раньше, могут работать с объектами `CommandBar` (см. раздел "Создание "старомодных" панелей инструментов"). Проблема заключается в том, что многие свойства и методы объекта `CommandBar` в Excel 2007 и Excel 2010 просто игнорируются. Например, панели инструментов и настраиваемые меню отображаются на вкладке ленты Надстройки (Add-Ins). Вследствие этого свойства, контролирующие размеры и положение панели инструментов, больше не работают. Также не поддерживаются "плавающие" панели инструментов. На приведенном ниже рисунке показаны настраиваемое меню и панель инструментов в Excel 2003 и те же меню и панель инструментов в Excel 2007 и Excel 2010. Несмотря на то что эти элементы пользовательского интерфейса остаются работоспособными в Excel 2007 и Excel 2010, они не соответствуют концепции пользовательского интерфейса, принятой разработчиками. Вполне естественно, что разработчики на языке VBA хотели бы вернуть те дни, когда они могли как угодно настраивать элементы пользовательского интерфейса.





В этой главе будет представлен простой пример разработки настраиваемой панели инструментов с помощью объекта `CommandBar` (см. раздел "Создание "старомодных" панелей инструментов"). За дополнительными сведениями о создании настраиваемых меню и панелей инструментов с помощью объекта `CommandBar` обращайтесь к предыдущим изданиям этой книги.

В Excel 2010 осталась возможность настройки контекстных меню, и эта тема подробнее рассматривается в главе 23.

Управление лентой с помощью VBA

У вас, наверное, уже возник вопрос: "Что можно сделать с лентой, используя VBA?" К сожалению, не слишком много.

Ниже представлен список операций, выполняемых с помощью VBA:

- определение, активизирован ли определенный элемент управления;
- определение, отображается ли определенный элемент управления;
- определение, был ли нажат определенный элемент управления (для кнопок и флажков);
- получение подписи элемента управления, экранной и расширенной подсказок (более подробное описание элемента управления);
- вывод изображения, связанного с элементом управления;
- выполнение определенной команды.

Ниже приведены действия по отношению к ленте, которые невозможно выполнить посредством VBA (даже при большом желании с вашей стороны):

- определение, выделена ли вкладка;
- активизирование определенной вкладки;
- добавление новой вкладки;
- добавление новой группы на вкладку;
- добавление нового элемента управления;
- удаление элемента управления;
- отключение элемента управления;
- сокрытие элемента управления.

Новинка

В Excel 2010 вносить изменения в ленту можно с помощью раздела Настройка ленты (Customize Ribbon), находящегося в диалоговом окне Параметры Excel (Excel Options). К сожалению, внести подобные изменения в ленту с помощью VBA невозможно.

Доступ к элементам управления на ленте

Как упоминалось ранее, на ленте Excel находится более чем 1700 элементов управления. Каждый из них имеет свое имя, которое можно использовать при доступе к этому элементу с помощью VBA.

Например, приведенный ниже оператор отображает окно сообщения, в котором показано состояние `Enabled` (Активизировано) элемента управления `ViewCustomViews`. (Этот элемент управления находится в группе команд, доступ к которой открывается после выполнения команд **Вид⇨Режимы просмотра книги** (View⇨Workbook Views).)

```
MsgBox Application.CommandBars.GetEnabledMso("ViewCustomViews")
```

Обычно этот элемент управления активизирован. Но если рабочая книга содержит таблицу (созданную путем выполнения команды **Вставка⇨Таблицы⇨Таблица** (Insert⇨Tables⇨Table)), элемент управления `ViewCustomViews` будет отключен.

Задача определения названия элемента управления выполняется вручную. Откройте вкладку **Настройка ленты** (Customize Ribbon) в окне, отображаемом после выполнения команды **Файл⇨Параметры Excel** (File⇨Excel Options). Найдите требуемый элемент

управления в левой части окна и установите над ним указатель мыши. После этого отобразится экранная подсказка (рис. 22.2).

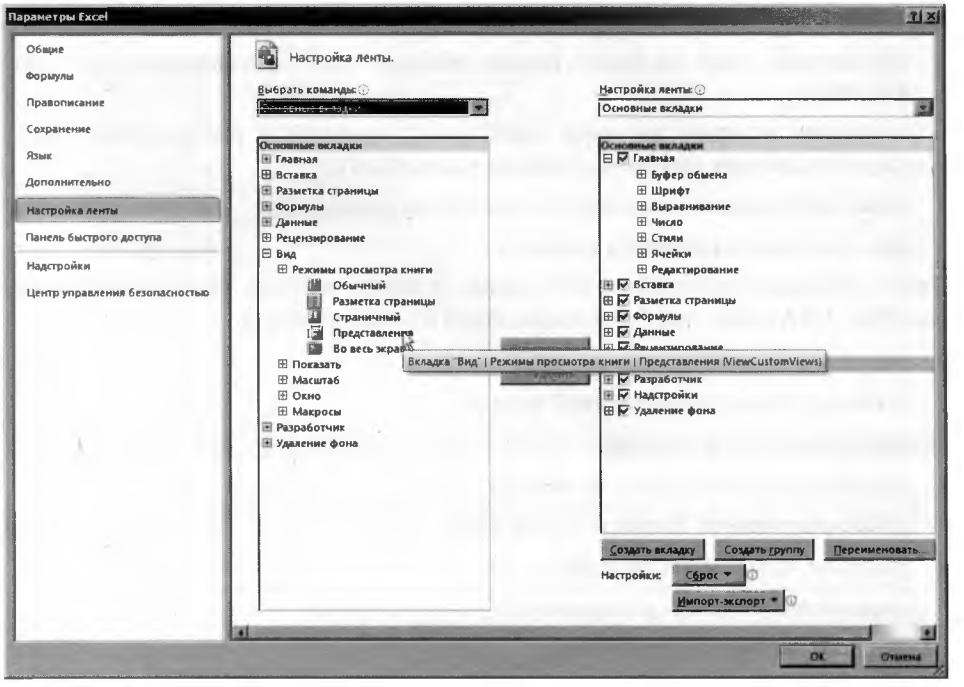


Рис. 22.2. Воспользуйтесь вкладкой Настройка ленты окна Параметры Excel для определения названия элемента управления

К сожалению, невозможно создать код на языке VBA, который циклически просматривал бы все элементы управления ленты, отображая их названия.

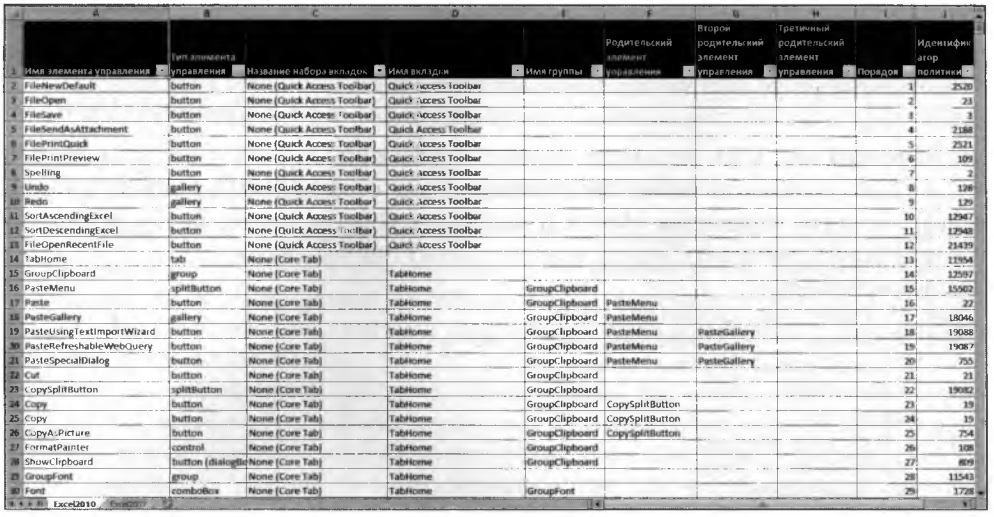
Компакт-диск

На прилагаемом компакт-диске находится рабочая книга, в которой содержатся названия всех элементов управления Excel. Здесь же можно найти дополнительные сведения о каждом элементе управления, включая тип элемента управления, название вкладки и группы. На рис. 22.3 показана часть содержимого этого файла (`ribbon control names.xlsx`).

Как работать с лентой

В предыдущем разделе приводился пример использования метода `GetEnabledMso` объекта `CommandBar`. Ниже приводится перечень методов объекта `CommandBar`, используемых для работы с лентой. Все они используют единственный аргумент `idMso`, который представляет название команды.

- `ExecuteMso` — вызывает элемент управления.
- `GetEnabledMso` — возвращает значение `True`, если активизирован указанный элемент управления.
- `GetImageMso` — возвращает графическое изображение элемента управления.



	A	B	C	D	E	F	G	H	I	J
1	Имя элемента управления	Тип элемента управления	Название набора вкладок	Имя вкладки	Имя группы	Родительский элемент управления	Второй родительский элемент управления	Третичный родительский элемент управления	Порядок	Идентификатор политики
2	FileNewDefault	button	None (Quick Access Toolbar)	Quick Access Toolbar					1	2520
3	FileOpen	button	None (Quick Access Toolbar)	Quick Access Toolbar					2	23
4	FileSave	button	None (Quick Access Toolbar)	Quick Access Toolbar					3	3
5	FileSendAsAttachment	button	None (Quick Access Toolbar)	Quick Access Toolbar					4	2188
6	FilePrintQuick	button	None (Quick Access Toolbar)	Quick Access Toolbar					5	2521
7	FilePrintPreview	button	None (Quick Access Toolbar)	Quick Access Toolbar					6	109
8	Spelling	button	None (Quick Access Toolbar)	Quick Access Toolbar					7	2
9	Undo	gallery	None (Quick Access Toolbar)	Quick Access Toolbar					8	128
10	Redo	gallery	None (Quick Access Toolbar)	Quick Access Toolbar					9	129
11	SortAscendingExcel	button	None (Quick Access Toolbar)	Quick Access Toolbar					10	12947
12	SortDescendingExcel	button	None (Quick Access Toolbar)	Quick Access Toolbar					11	12948
13	FileOpenRecentFile	button	None (Quick Access Toolbar)	Quick Access Toolbar					12	21439
14	TabHome	tab	None (Core Tab)						13	11954
15	GroupClipboard	group	None (Core Tab)	TabHome					14	12597
16	PasteMenu	splitButton	None (Core Tab)	TabHome	GroupClipboard				15	15502
17	Paste	button	None (Core Tab)	TabHome	GroupClipboard	PasteMenu			16	22
18	PasteGallery	gallery	None (Core Tab)	TabHome	GroupClipboard	PasteMenu			17	18046
19	PasteUsingTextImportWizard	button	None (Core Tab)	TabHome	GroupClipboard	PasteMenu	PasteGallery		18	19088
20	PasteRefreshableWebQuery	button	None (Core Tab)	TabHome	GroupClipboard	PasteMenu	PasteGallery		19	19087
21	PasteSpecialDialog	button	None (Core Tab)	TabHome	GroupClipboard	PasteMenu	PasteGallery		20	755
22	Cut	button	None (Core Tab)	TabHome	GroupClipboard				21	21
23	CopySplitButton	splitButton	None (Core Tab)	TabHome	GroupClipboard				22	19082
24	Copy	button	None (Core Tab)	TabHome	GroupClipboard	CopySplitButton			23	19
25	Copy	button	None (Core Tab)	TabHome	GroupClipboard	CopySplitButton			24	19
26	CopyAsPicture	button	None (Core Tab)	TabHome	GroupClipboard	CopySplitButton			25	754
27	FormatPainter	control	None (Core Tab)	TabHome	GroupClipboard				26	108
28	ShowClipboard	button (dialogBo	None (Core Tab)	TabHome	GroupClipboard				27	809
29	GroupFont	group	None (Core Tab)	TabHome					28	11543
30	Font	comboBox	None (Core Tab)	TabHome	GroupFont				29	1728

Рис. 22.3. Рабочая книга, отображающая сведения о каждом элементе управления лентой

- `GetLabelMso` — возвращает подпись для элемента управления.
- `GetPressedMso` — возвращает значение `True`, если выбран указанный элемент управления (применяется для работы с кнопками и флажками).
- `GetScreentipMso` — возвращает экранную подсказку для элемента управления (текст, отображаемый поверх элемента управления).
- `GetSupertipMso` — возвращает подробную подсказку для элемента управления (описание элемента управления, которое отображается после установки указателя мыши над элементом управления).

Некоторые из этих методов практически бесполезны. Например, для чего программисту на VBA определять экранную подсказку для элемента управления? По моему мнению, этого не требуется.

Следующий оператор VBA активизирует панель выделения (Selection pane) — свойство, появившееся в Excel 2007, которое облегчает выделение объектов на рабочем листе.

```
Application.CommandBars.ExecuteMso("SelectionPane")
```

Представленный ниже оператор отображает диалоговое окно Специальная вставка (Paste Special).

```
Application.CommandBars.ExecuteMso("PasteSpecialDialog")
```

С помощью следующего оператора можно определить, отображается ли на экране панель формул. По сути, этот оператор проверяет, установлен ли флажок Строка формул (Formula Bar), находящийся в группе Показать вкладки ленты Вид.

```
MsgBox Application.CommandBars.GetPressedMso("ViewFormulaBar")
```

Обратите внимание на то, что код не может непосредственно отобразить или скрыть панель формул путем установки (либо отмены установки) флажка Строка формул. Для выполнения этой задачи воспользуйтесь свойством `DisplayFormulaBar` объекта `Application`.

```
Application.DisplayFormulaBar = True
```

Приведенный ниже оператор возвращает значение `True`, если активизирован элемент управления **Объединить и поместить в центре** (Merge & Center). (Этот элемент управления отключен, если лист защищен или активная ячейка находится в таблице.)

```
MsgBox Application.CommandBars.GetEnabledMso("MergeCenter")
```

Следующий код VBA добавляет элемент управления ActiveX `Image` на активный лист, а также использует метод `GetImageMso` для отображения картинки, соответствующей элементу управления **Найти и выделить** (Find & Select), который находится в группе **Редактирование** (Editing) вкладки денты **Главная** (Home).

```
Sub ImageOnSheet()
   Dim MyImage As OLEObject
   Set MyImage = ActiveSheet.OLEObjects.Add _
      (ClassType:="Forms.Image.1", _
      Left:=50, _
      Top:=50)
   With MyImage.Object
      .AutoSize = True
      .BorderStyle = 0
      .Picture = Application.CommandBars. _
         GetImageMso("FindDialog", 32, 32)
   End With
End Sub
```

Для отображения значка ленты в области элемента управления `Image` (под названием `Image1`), входящего в состав пользовательской формы, воспользуйтесь следующей процедурой.

```
Private Sub UserForm_Initialize()
   With Image1
      .Picture = Application.CommandBars. _
         GetImageMso("FindDialog", 32, 32)
      .AutoSize = True
   End With
End Sub
```

Активизация вкладки

Непосредственного способа активизации вкладок ленты с помощью VBA не существует. Если необходимо сделать это, воспользуйтесь методом `SendKeys`. Этот метод имитирует нажатия клавиш. Для активизации вкладки ленты **Главная** (Home) нужно нажать клавишу <Alt>, а затем — <H>. Нажатие этих клавиш приводит к отображению на ленте подсказок клавиш. Для сокрытия этих подсказок нажмите клавишу <F6>. Подводя итоги, скажем, что для активизации вкладки **Главная** (Home) используется следующий оператор:

```
Application.SendKeys "%h{F6}"
```

Ниже приведены аргументы метода `SendKeys`, используемые для активизации других вкладок:

- **Вставка (Insert):** `"%n{F6}"`;
- **Разметка страницы (Page Layout):** `"%p{F6}"`;
- **Формулы (Formulas):** `"%m{F6}"`;
- **Данные (Data):** `"%a{F6}"`;
- **Рецензирование (Review):** `"%r{F6}"`;

- **Вид (View):** `"%w{F6}"`;
- **Разработчик (Developer):** `"%l{F6}"`;
- **Надстройки (Add-Ins):** `"%x{F6}"`.

Настройка ленты

С помощью VBA невозможно изменять ленту произвольным образом. Для этого нужно написать код RibbonX и включить его в файл рабочей книги. Данный код создается в XML-редакторе. Затем можно создать *процедуру обратного вызова* VBA. Она представляет собой макрос VBA, который вызывается после активизации настраиваемого элемента управления ленты.

RibbonX — это, по сути, XML-код, описывающий элементы управления, которые отображаются на ленте. Данный код определяет их внешний вид, а также действия, происходящие при активизации этих элементов управления. Подробное рассмотрение кода RibbonX выходит за рамки книги. В следующих разделах рассматривается несколько простых примеров, которые иллюстрируют то, что происходит при модификации пользовательского интерфейса Excel.

Перекрестная ссылка

Дополнительные сведения о структуре файла Excel можно найти в главе 4. В данном разделе описан порядок просмотра данных файла рабочей книги `XLSX`.

Простой пример кода RibbonX

В этом разделе приводится пошаговая инструкция по изменению ленты Excel. В данном примере создается новая группа команд ленты (под названием Custom (Пользовательская)), которая находится на вкладке Данные (Data). В новой группе ленты также создаются две новые кнопки под названием Hello World (Здравствуй мир) и Goodbye World (Прощай мир). Щелчок на каждой из этих кнопок вызывает соответствующий макрос VBA.

Примечание

Приведенные ниже инструкции весьма трудоемкие, а их реализация на практике чревата появлением ошибок. Поэтому большинство разработчиков не используют данный метод на практике.

Сохранение изменений интерфейса пользователя

Пользователи Excel 2010 могут вносить изменения в ленту и панель быстрого доступа. В частности, не составляет особого труда добавление команд на панель быстрого доступа либо ленту. Каким же образом Excel отслеживает подобные изменения?

Сведения об изменениях, внесенных в панель быстрого доступа, хранятся в файле `Excel.officeUI`. Местоположение этого файла различно. Например, в моей системе он находится в такой папке:

`C:\Users\<имя_пользователя>\AppData\Local\Microsoft\Office`

Дополнительные сведения о файле `Excel.officeUI` можно найти в главе 4.

Использование метода SendKeys далеко не всегда дает хороший результат. Например, если вызвать на выполнение предыдущий пример во время отображения пользовательской формы, коды нажатых клавиш будут пересланы форме UserForm, а не ленте.

Просмотр ошибок

Прежде чем выполнять любые настройки ленты, следует включить отображение ошибок, возникающих при использовании метода RibbonX. Выполните команду Файл⇨Параметры Excel (File⇨Excel Options) для открытия диалогового окна Параметры Excel, затем выберите вкладку Дополнительно (Advanced) и перейдите в раздел Дополнительно (General). Здесь установите флажок Показывать ошибки интерфейса пользователя надстроек (Show Add-in User Interface Errors).

После выбора этой настройки будут отображаться ошибки RibbonX (в случае их наличия) при открытии рабочей книги — очень полезная возможность на этапе отладки.

Для создания рабочей книги, содержащей код RibbonX, который изменяет ленту, выполните следующие действия.

1. Создайте новую рабочую книгу Excel, вставьте модуль VBA и введите код двух процедур обратного вызова.

   Ниже приводится код двух процедур, которые вызываются после щелчка мышью на кнопках.

```
Sub HelloWorld(control As IRibbonControl)
   MsgBox "Здравствуй мир"
End Sub
Sub GoodbyeWorld(control As IRibbonControl)
   ThisWorkbook.Close
End Sub
```

2. Сохраните рабочую книгу, присвоив ей имя ribbon modification.xlsm.
3. Закройте рабочую книгу.
4. Активизируйте папку, которая содержит файл ribbon modification.xlsm, и создайте папку customUI.
5. Находясь в этой папке, воспользуйтесь текстовым редактором (например, Windows Notepad) для создания текстового файла customUI.xml, который содержит следующий XML-код RibbonX.

```
<customUI xmlns="http://schemas.microsoft.com/office/2006/01/ _
   customui">
<ribbon>
<tabs>
<tab idMso="TabData">
   <group id="GroupI" label="Custom">
      <button id="Button1"
         label="Здравствуй мир"
         size="normal"
         onAction="HelloWorld"
         imageMso="HappyFace" />
      <button id="Button2"
         label="Прощай мир"
         size="normal"
         onAction="GoodbyeWorld"
         imageMso="DeclineInvitation" />
```

```
        </group>
    </tab>
    </tabs>
    </ribbon>
    </customUI>
```

6. С помощью Проводника Windows добавьте расширение `.zip` к файлу `ribbon modification.xlsm`. После этого файл получит название `ribbon modification.xlsm.zip`.
7. Перетащите папку `customUI`, созданную в п. 4, на файл `ribbon modification.xlsm.zip`. Windows трактует ZIP-файлы в качестве папок, поэтому операции перетаскивания разрешаются.
8. Дважды щелкните на файле `ribbon modification.xlsm.zip` для его открытия. На рис. 22.4 показано содержимое ZIP-файла. Как видите, файл включает множество папок.

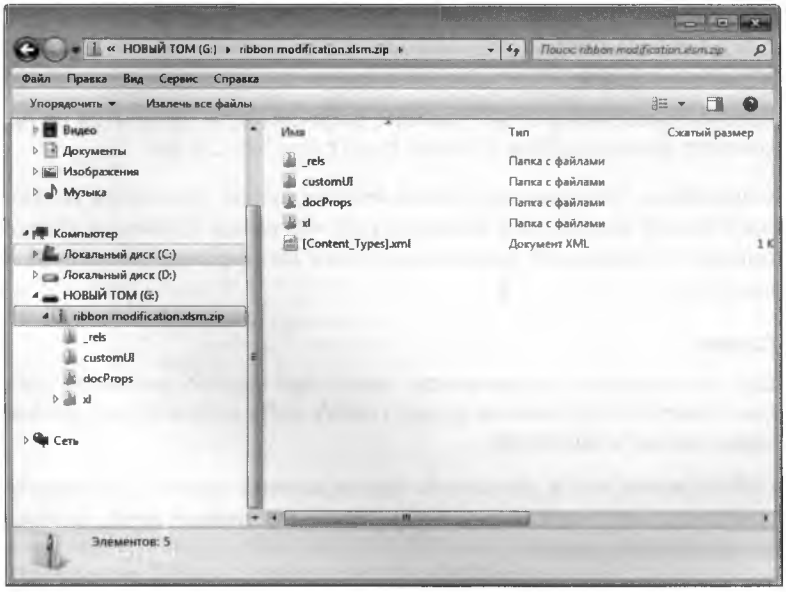


Рис. 22.4. Рабочая книга Excel, представленная в виде ZIP-файла

9. Дважды щелкните мышью на папке `_rels`, которая входит в состав ZIP-файла.

   В этой папке находится один файл с расширением `.rels`.
10. Перетащите файл с расширением `.rels` за переделы ZIP-файла (например, на рабочий стол).
11. Откройте файл с расширением `.rels` (по сути, это XML-файл) в окне текстового редактора — например, в Блокноте.
12. Добавьте в файл с расширением `.rels` следующую строку кода — перед тегом `</Relationships>`.

```
<Relationship Type="http://schemas.microsoft.com/office/2006/
   relationships/ui/extensibility" Target= _
```

```
"/customUI/customUI.xml"
Id="12345" />
```

13. Сохраните файл с расширением `.rels`, затем перетащите его обратно в область ZIP-файла, заменив прежнюю версию этого файла.
14. Удалите расширение `.zip`. После этого возвращается прежнее название файла: `ribbon modification.xlsm`.

Откройте рабочую книгу в Excel. Если все было сделано правильно, вы увидите новую группу с двумя новыми кнопками (Hello World и Goodbye World), находящуюся на вкладке Данные (Data), как показано на рис. 22.5.

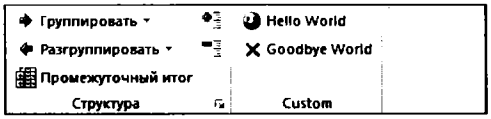


Рис. 22.5. С помощью кода RibbonX была создана новая группа, содержащая две кнопки

Компакт-диск

Рассматриваемая в этом примере рабочая книга находится на прилагаемом компакт-диске в файле `ribbon modification.xlsm`.

Обратите внимание, что новая группа на вкладке ленты появляется только в том случае, когда код RibbonX находится в активной рабочей книге. Именно в этом заключается основное отличие от изменений пользовательского интерфейса, выполненных в предыдущих версиях Excel.

Совет

Для отображения измененной ленты при любой активной рабочей книге преобразуйте последнюю в надстройку либо добавьте код RibbonX в персональную книгу макросов.

Если вы обнаружите, что в результате предпринятых усилий желаемого изменения ленты не произошло, не отчаивайтесь. Существует инструментарий, который позволит облегчить этот нелегкий процесс.

Усовершенствование простого примера кода RibbonX

В этом разделе приводится пошаговая инструкция по внесению изменений в ленту, которые аналогичны изменениям, описанным в предыдущем разделе. В данном случае использовалось приложение Custom UI Editor для Microsoft Office. Эта программа позволяет не только создавать код RibbonX, но и проверять ранее созданный код. Она также устраняет необходимость в выполнении сложных манипуляций с файлами. Наконец, эта программа в состоянии генерировать объявления процедур обратного вызова VBA, которые можно скопировать и вставить в модуль VBA.

Загрузите свободно распространяемую программу Custom UI Editor для Microsoft Office по такому адресу:

`http://openxmldeveloper.org/articles/customuieditor.aspx`

Для добавления на ленту новой группы и кнопок с помощью Custom UI Editor (как описано в предыдущем разделе) выполните следующие действия.

1. Откройте Excel, создайте новую рабочую книгу и сохраните ее в формате книги с поддержкой макросов (файл XLSM).
2. Закройте рабочую книгу.
3. Запустите программу Custom UI Editor для Microsoft Office.
4. Выполните команду File⇨Open (Файл⇨Открыть) и найдите рабочую книгу, сохраненную в п. 1.
5. Выберите пункты меню Insert⇨Office 2007 Custom UI Part (Вставка⇨Настраиваемый компонент интерфейса пользователя Office 2007).

   Благодаря выбору этих пунктов меню обеспечивается совместимость созданного файла с Excel 2007 и Excel 2010.
6. Введите код RibbonX, как показано на рис. 22.6.

Рис. 22.6. Приложение Custom UI Editor для Microsoft Office

7. Щелкните на кнопке верификации для выполнения проверки на наличие ошибок.
8. Щелкните на кнопке Generate Callbacks (Генерировать процедуры обратного вызова) и скопируйте появившийся на экране код.

   Приложение Custom UI Editor сгенерирует две процедуры обратного вызова VBA (см. рис. 22.7). Выделите и скопируйте этот код; позднее его можно вставить в модуль VBA для рабочей книги.
9. Перейдите на левую панель и щелкните на узле `customUI.xml`.
10. Выполните команду File⇨Save (Файл⇨Сохранить), затем выберите пункты меню File⇨Close (Файл⇨Закрыть).
11. Запустите Excel и откройте рабочую книгу.
12. Нажмите комбинацию клавиш <Alt+11> для активизации VB Editor.

13. Вставьте модуль VBA, а затем — код, который был скопирован в п. 8.
14. Добавьте оператор `MsgBox` в каждую процедуру обратного вызова VBA, позволяющий проверять факт их выполнения.

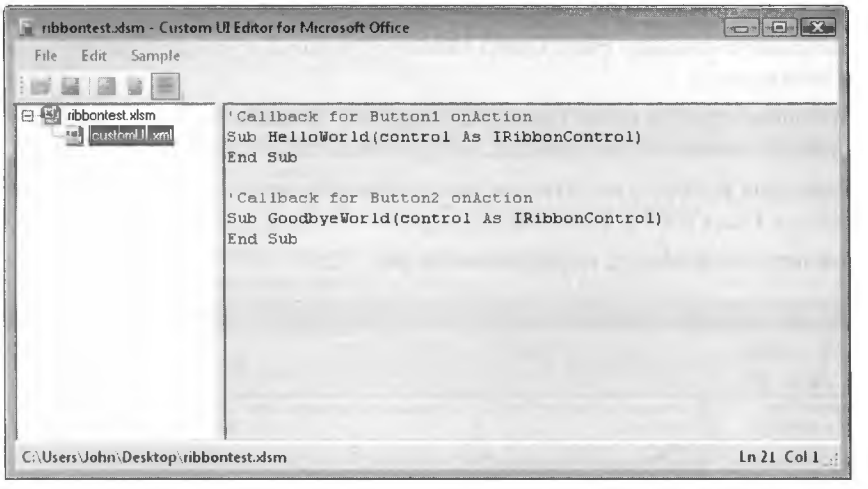


Рис. 22.7. Приложение Custom UI Editor сгенерировало две процедуры обратного вызова VBA

Как видите, использование приложения Custom UI Editor намного проще, чем написание кода RibbonX вручную.

Компонент CUSTOM UI

В п. 5 приведенной выше инструкции был выбран пункт меню Office 2007 CustomUI Part. В результате обеспечивается совместимость с Excel 2007 и Excel 2010. В меню Insert также можно выбрать пункт Office 2010 Custom UI Part. Если остановиться на этом пункте меню и поместить код RibbonX в компонент Office 2010 Custom UI Part, рабочая книга будет несовместима с Excel 2007.

Если создаваемое вами приложение не использует команды, которые уникальны для Excel 2010, лучше всего выбрать компонент Office 2007 Custom UI Part. Также имейте в виду, что единственный файл может включать компоненты, предназначенные для Office 2007 и Office 2010. Использовать два компонента целесообразно в том случае, когда для создания настраиваемого интерфейса пользователя используется код RibbonX, специфичный для каждой версии Excel. Например, код RibbonX может вызывать команду из режима просмотра Backstage, который появился в Excel 2010. Поскольку в Excel 2007 этот режим просмотра отсутствует, понадобится еще одна версия кода, которая вызывает команду из меню кнопки Office в Excel 2007.

Обратите внимание на первый оператор кода RibbonX, который изменяется в случае использования компонента Office 2010 Custom UI Part. В данном случае код должен ссылаться на следующее пространство имен.

```
<customUI xmlns='http://schemas.microsoft.com/office/2009/07/customui'>
```

Учтите, что в случае использования некорректного тега `customUI` приложение Custom UI Editor отобразит соответствующее сообщение в процессе верификации кода.

Процедуры обратного вызова VBA

Как вы помните, рабочая книга включает две процедуры VBA: `HelloWorld` и `GoodbyeWorld`. Имена этих процедур соответствуют параметрам `onAction` в коде RibbonX. Именно с помощью параметра `onAction` код RibbonX может быть связан с кодом VBA.

Обе процедуры VBA содержат аргумент `control`, который представляет собой объект `IRibbonControl`. Этот объект включает три свойства, доступ к которым обеспечивается с помощью кода VBA.

- `Context`. Дескриптор активного окна, содержащий ленту, которая была активизирована с помощью обратного вызова. Например, воспользуйтесь следующим выражением для получения имени рабочей книги, которая включает код RibbonX:
  `control.Context.Caption`
- `Id`. Включает имя элемента управления, указанного с помощью его параметра `Id`.
- `Tag`. Содержит произвольный лист, который связан с элементом управления.

Процедуры обратного вызова VBA могут иметь произвольную степень сложности.

Файл `.rels`

После вставки файла, содержащего код RibbonX, эффект достигается после установки связи между файлами документа и настроек. Эти связи создаются с помощью XML-кода и хранятся в файле с расширением `.rels`, который находится в папке `_rels`. Ниже представлена связь для примера, представленного в предыдущем разделе.

```
<Relationship Type="http://schemas.microsoft.com/office/2006/
relationships/ui/extensibility" Target="/customUI/customUI.xml"
   Id="12345" />
```

Параметр `Target` указывает на файл `customUI.xml`, который содержит код RibbonX. Значение параметра `Id` — любая строка, выбранная случайным образом. Таким образом, эта строка может содержать любую информацию, а ее длина выбирается из соображений уникальности файла (т.е. длина должна быть настолько большой, чтобы не нашелся другой тег `<Relationship>`, использующий тот же параметр `Id`).

Код RibbonX

А теперь рассмотрим более сложный вопрос — создание XML-кода, который выполняет изменение пользовательского интерфейса. Как отмечалось ранее, в этой книге не рассматриваются методы написания кода RibbonX. Все ограничивается несколькими примерами, и для получения другой информации следует обращаться к другим источникам.

Использование изображений `imageMso`

В Microsoft Office 2010 поддерживается более 1000 именованных изображений, которые связаны с различными командами. Если вы знаете имя изображения, можете связать это изображение с пользовательскими элементами управления ленты.

На следующем рисунке показана рабочая книга, которая содержит имена всех изображений `imageMso`. При просмотре имен изображений можно видеть одновременно до 50 изображений (маленького либо среднего размеров), начиная с имени изображения, находящегося в активной ячейке. Эта рабочая книга под именем `mso image browser.xlsm` находится на прилагаемом к книге компакт-диске.

Эти изображения могут использоваться в составе элемента управления `Image` в окне `UserForm`. Находящийся ниже оператор назначает изображение `imageMso` под именем `ReviewAcceptChanges` свойству `Picture` элемента управления `UserForm Image` под именем `Image1`. Размер изображения — 32×32 пикселя.

```
Image1.Picture = Application.CommandBars.
   GetImageMso("ReviewAcceptChange", 32, 32)
```

Примечание

Вы, наверное, будете удивлены, узнав, что параметр `imageMso` определяет более 1000 значков, которые можно связать с каждым из элементов управления ленты, доступ к которым осуществляется по имени. Дополнительные сведения по этой теме можно найти во врезке "Использование изображений `imageMso`".

Еще один пример кода RibbonX

В этом разделе рассматривается еще один пример использования кода RibbonX для изменения пользовательского интерфейса. Рассматриваемая в разделе рабочая книга соз-

ет новую группу на вкладке Разметка страницы (Page Layout), а также добавляет ажок, который включает отображение разрывов.

Примечание

Несмотря на то что в Excel находится более 1700 команд, отсутствует команда, предназначенная для отображения разрывов страниц. Помимо печати и предварительного просмотра страниц, единственный способ скрыть разрывы страниц — использование диалогового окна Параметры Excel (Excel Options). Поэтому этот пример имеет также практическую ценность.

Этот пример довольно сложен, поскольку он требует, чтобы новый элемент управления лентой был синхронизирован с активным листом. Например, если был активизирован рабочий лист, который не отображает разрывы страниц, флажок должен быть установленным. Более того, разрывы страниц не могут отображаться на листе диаграммы, поэтому флажок должен быть сброшен, если активизируется лист диаграммы.

од RibbonX

Следующий код RibbonX добавляет новую группу (с флажком) на вкладку Разметка раницы (Page Layout).

```
ustomUI
  xmlns="http://schemas.microsoft.com/office/2006/01/customui"
  onLoad="Initialize">
ibbon>
abs>
ab idMso="TabPageLayoutExcel">
  <group id="Group1" label="Custom">
     <checkBox id="Checkbox1"
        label="Page Breaks"
        onAction="TogglePageBreakDisplay"
        getPressed="GetPressed"
        getEnabled="GetEnabled"/>
  </group>
/tab>
/tabs>
/ribbon>
/customUI>
```

Этот код RibbonX ссылается на четыре процедуры обратного вызова VBA, каждая из оторых описана ниже:

- `Initialize` — вызывается при открытии рабочей книги;
- `TogglePageBreakDisplay` — вызывается после щелчка мышью на флажке;
- `GetPressed` — вызывается при деактивизации элемента управления (нажимается клавиша);
- `GetEnabled` — вызывается при деактивизации элемента управления (активизируется другой лист).

На рис. 22.8 показан новый элемент управления.

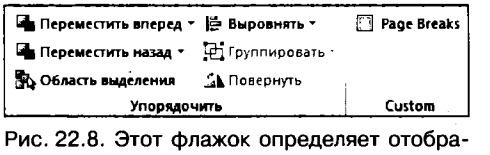


Рис. 22.8. Этот флажок определяет отображение разрывов страниц на активном листе

Код VBA

Тег <CustomUI> включает параметр onLoad, который определяет процедуру обратного вызова VBA, код которой приводится ниже.

```
Public MyRibbon As IRibbonUI
Sub Initialize(Ribbon As IRibbonUI)
'  Вызывается при загрузке рабочей книги
   Set MyRibbon = Ribbon
End Sub
```

Процедура Initialize создает объект IRibbonUI, который называется MyRibbon. Обратите внимание на то, что MyRibbon — это глобальная переменная, поэтому доступ к ней возможен со стороны других процедур в этом модуле.

Затем была создана простая процедура обработки событий, которая вызывается в случае активизации рабочего листа. Эта процедура, находящаяся в модуле кода ThisWorkbook (ЭтаКнига), вызывает процедуру CheckPageBreakDisplay.

```
Private Sub Workbook_SheetActivate(ByVal Sh As Object)
   Call CheckPageBreakDisplay
End Sub
```

Процедура CheckPageBreakDisplay *отменяет установку* флажка. Другими словами, она разрушает все данные, связанные с этим элементом управления.

```
Sub CheckPageBreakDisplay()
'  Вызывается при активизации листа
   MyRibbon.InvalidateControl ("Checkbox1")
End Sub
```

После отмены установки флажка вызываются процедуры GetPressed и GetEnabled.

```
Sub GetPressed(control As IRibbonControl, ByRef returnedVal)
'  Вызывается после деактивизации элемента управления
   On Error Resume Next
   returnedVal = ActiveSheet.DisplayPageBreaks
End Sub
Sub GetEnabled(control As IRibbonControl, ByRef returnedVal)
'  Вызывается после отмены установки флажка
   returnedVal = TypeName(ActiveSheet) = "Worksheet"
End Sub
```

Обратите внимание на то, что аргумент returnedVal передается по ссылке (ByRef). Это означает, что код может изменять значение — именно это и происходит в данном случае. При выполнении процедуры GetPressed переменной returnedVal присваивается значение, соответствующее состоянию свойства DisplayPageBreaks активного листа. В результате параметру Pressed элемента управления присваивается значение True (если отображаются разрывы страниц). При этом флажок установлен. Если же значение этого параметра False, флажок не устанавливается.

В процедуре GetEnabled переменной returnedVal присваивается значение True, если в качестве активного листа выбран рабочий лист (не лист диаграммы). Таким образом, элемент управления активизируется только в том случае, когда в качестве активного листа выбран рабочий лист.

В качестве вспомогательной процедуры VBA выбрана процедура onAction под названием TogglePageBreakDisplay, которая выполняется в том случае, когда пользователь устанавливает или снимает флажок.

```
Sub TogglePageBreakDisplay(control As IRibbonControl, _
   pressed As Boolean)
'  Выполняется, если флажок установлен
   On Error Resume Next
   ActiveSheet.DisplayPageBreaks = pressed
End Sub
```

Аргументу `Pressed` присваивается значение `True`, если флажок установлен, и `False` — если он не установлен. Также код соответствующим образом устанавливает значение свойства `DisplayPageBreaks`.

Компакт-диск

Рассматриваемая в этом разделе рабочая книга находится на прилагаемом компакт-диске (в файле `page break display.xlsm`). Здесь же находится версия этой рабочей книги, реализованная в виде надстройки (`page break display add-in.xlam`), которая создает новую команду пользовательского интерфейса. Эта версия доступна для всех рабочих книг. Надстройка использует модуль класса для отслеживания событий активизации листа для всех рабочих книг. Дополнительные сведения о событиях можно найти в главе 19, а о модулях классов — в главе 29.

Демонстрация возможностей элементов управления ленты

На рис. 22.9 показана пользовательская вкладка ленты My Stuff, включающая четыре группы элементов управления. В этом разделе будет кратко описан код RibbonX, а также процедуры обратного вызова VBA.

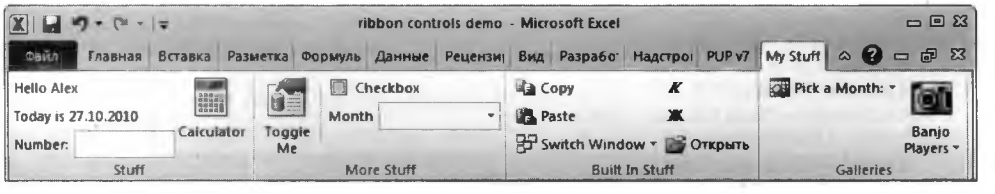


Рис. 22.9. Новая вкладка ленты с четырьмя группами элементов управления

Компакт-диск

Рассматриваемая в этом разделе рабочая книга находится на прилагаемом к книге компакт-диске в файле `ribbon controls demo.xlsm`.

Создание новой вкладки

Ниже показан код RibbonX, который создает новую вкладку.

```
<ribbon>
   <tabs>
      <tab id="CustomTab" label="My Stuff">
   </tabs>
</ribbon>
```

Совет

Для создания "минимального" пользовательского интерфейса используется атрибут `startFromScratch` тега `<ribbon>`. Если значение этого атрибута равно `True`, скрываются все встроенные вкладки.

```
<ribbon startFromScratch="true" >
```

Создание группы ленты

Код из примера `ribbon controls demo.xlsm` создает четыре группы на вкладке My Stuff. Ниже представлен код, создающий эти четыре группы.

```
<group id="Group1" label="My Stuff">
</group>
<group id="Group2" label="More Stuff">
</group>
<group id="Group3" label="Built In Stuff">
</group>
<group id="Group4" label="Galleries">
</group>
```

Эти пары тегов (`<group>`, `</group>`) находятся между тегами `<tab>` и `</tab>`, которые создают новую вкладку.

Создание элементов управления

Ниже представлен код RibbonX, который создает элементы управления в первой группе (Stuff), показанной на рис. 22.10. Обратите внимание на то, что элементы управления определены в первом наборе тегов `<group>`.

```
<group id="Group1" label="My Stuff">
   <labelControl id="Label1" getLabel="getLabel1" />
   <labelControl id="Label2" getLabel="getLabel2" />

   <editBox id="EditBox1"
      showLabel="true"
      label="Number:"
      onChange="EditBox1_Change"/>
   <button id="Button1"
      label="Calculator"
      size="large"
      onAction="ShowCalculator"
      imageMso="Calculator" />
</group>
```

Для каждого из двух элементов управления `Label` (подпись) используется связанная с ним процедура обратного вызова VBA (`getLabel1` и `getLabel2`). Ниже представлен код этих процедур.

```
Sub getLabel1(control As IRibbonControl, ByRef returnedVal)
   returnedVal = "Hello " & Application.UserName
End Sub
Sub getLabel2(control As IRibbonControl, ByRef returnedVal)
   returnedVal = "Today is " & Date
End Sub
```

Непосредственно после загрузки кода RibbonX вызываются на выполнение эти две процедуры, а заголовки элементов управления `Label` динамически обновляются значениями имени пользователя и даты.

Элемент управления `editBox` включает процедуру обратного вызова `onChange` под названием `EditBox1_Change`, которая вычисляет квадратный корень от введенного

числа (или отображает сообщение об ошибке, если квадратный корень не может быть вычислен). Ниже представлен код этой процедуры.

```
Sub EditBox1_Change(control As IRibbonControl, text As String)
   Dim squareRoot As Double
   On Error Resume Next
   squareRoot = Sqr(text)
   If Err.Number = 0 Then
      MsgBox "The square root of " & text & " _
         is: " & squareRoot
   Else
      MsgBox "Enter a positive number.", vbCritical
   End If
End Sub
```

Последний элемент управления группы `Stuff` — обычная кнопка. Связанный с ней параметр `onAction` вызывает процедуру VBA под именем `ShowCalculator`, которая использует функцию VBA `Shell` для отображения калькулятора Windows.

```
Sub ShowCalculator(control As IRibbonControl)
   On Error Resume Next
   Shell "calc.exe", vbNormalFocus
   If Err.Number <> 0 Then MsgBox "Can't start calc.exe"
End Sub
```

На рис. 22.11 показаны элементы управления из второй группы, которая называется `More Stuff`.

Рис. 22.10. Группа ленты, включающая четыре элемента управления

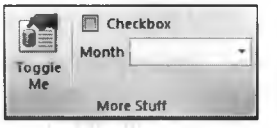


Рис. 22.11. Три элемента управления в настраиваемой группе ленты

Ниже представлен код RibbonX для второй группы.

```
<group id="Group2" label="More Stuff">
   <toggleButton id="ToggleButton1"
      size="large"
      imageMso="FileManageMenu"
      label="Toggle Me"
      onAction="ToggleButton1_Click" />
   <separator id="sep1" />
   <checkBox id="Checkbox1"
      label="Checkbox"
      onAction="Checkbox1_Change"/>
   <comboBox id="Combo1"
      label="Month"
      onChange="Combo1_Change">
      <item id="Month1" label="January" />
      <item id="Month2" label="February"/>
      <item id="Month3" label="March"/>
      <item id="Month4" label="April"/>
      <item id="Month5" label="May"/>
      <item id="Month6" label="June"/>
      <item id="Month7" label="July"/>
      <item id="Month8" label="August"/>
      <item id="Month9" label="September"/>
      <item id="Month10" label="October"/>
```

```
      <item id="Month11" label="November"/>
      <item id="Month12" label="December"/>
   </comboBox>
</group>
```

Эта группа включает следующие элементы управления: `toggleButton`, разделитель, `checkBox` и `comboBox`. Эти элементы управления просты в применении и не требуют дополнительных объяснений. За исключением разделителя (вставляет вертикальную линию), для каждого элемента управления существует связанная с ним процедура обратного вызова, которая просто отображает состояние этого элемента.

```
Sub ToggleButton1_Click(control As IRibbonControl, _
   ByRef returnedVal)
   MsgBox "Toggle value: " & returnedVal
End Sub
Sub Checkbox1_Change(control As IRibbonControl, _
   pressed As Boolean)
   MsgBox "Checkbox value: " & pressed
End Sub
Sub Combo1_Change(control As IRibbonControl, text As String)
   MsgBox text
End Sub
```

Примечание

Элемент управления `comboBox` также принимает текст, введенный пользователем. Если нужно ограничить количество вариантов выбора, предлагаемых этим элементом, воспользуйтесь элементом управления `dropDown`.

Элементы управления в третьей группе являются встроенными (рис. 22.12). Для включения встроенного элемента управления в пользовательскую группу следует знать его имя (значение параметра `idMso`).

Ниже представлен код RibbonX третьей группы ленты.

```
<group id="Group3" label="Built In Stuff">
   <control idMso="Copy" label="Copy" />
   <control idMso="Paste" label="Paste" enabled="true" />
   <control idMso="WindowSwitchWindowsMenuExcel"
      label="Switch Window" />
   <control idMso="Italic" />
   <control idMso="Bold" />
   <control idMso="FileOpen" />
</group>
```

Для этих элементов управления отсутствуют процедуры обратного вызова, поскольку данные элементы выполняют стандартные действия.

На рис. 22.13 показана последняя группа элементов управления, которая состоит из двух коллекций.

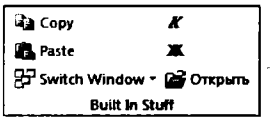


Рис. 22.12. В этой группе находятся встроенные элементы управления

Рис. 22.13. Эта группа включает две коллекции

Ниже приводится код RibbonX, определяющий эти коллекции.

```
group id="Group4" label="Galleries">
  <gallery id="Gallery1"
     imageMso="ViewAppointmentInCalendar"
     label="Pick a Month:"
     columns="2" rows="6"
     onAction="MonthSelected" >
  <item id="January" label="January" _
     imageMso="QuerySelectQueryType"/>
  <item id="February" label="February" _
     imageMso="QuerySelectQueryType"/>
  <item id="March" label="March" _
     imageMso="QuerySelectQueryType"/>
  <item id="April" label="April" _
     imageMso="QuerySelectQueryType"/>
  <item id="May" label="May" _
     imageMso="QuerySelectQueryType"/>
  <item id="June" label="June" _
     imageMso="QuerySelectQueryType"/>
  <item id="July" label="July" _
     imageMso="QuerySelectQueryType"/>
  <item id="August" label="August" _
     imageMso="QuerySelectQueryType"/>
  <item id="September" label="September" _
     imageMso="QuerySelectQueryType"/>
  <item id="October" label="October" _
     imageMso="QuerySelectQueryType"/>
  <item id="November" label="November" _
     imageMso="QuerySelectQueryType"/>
  <item id="December" label="December" _
     imageMso="QuerySelectQueryType"/>
     <button id="Today"
        label="Today..."
        imageMso="ViewAppointmentInCalendar"
        onAction="ShowToday"/>
     </gallery>
     <gallery id="Gallery2"
        label="Banjo Players"
        size="large"
        columns="4"
        itemWidth="100" itemHeight="125"
        imageMso= "Camera"
        onAction="OnAction">
        <item id="bp01" image="bp01" />
        <item id="bp02" image="bp02" />
        <item id="bp03" image="bp03" />
        <item id="bp04" image="bp04" />
        <item id="bp05" image="bp05" />
        <item id="bp06" image="bp06" />
        <item id="bp07" image="bp07" />
        <item id="bp08" image="bp08" />
        <item id="bp09" image="bp09" />
        <item id="bp10" image="bp10" />
        <item id="bp11" image="bp11" />
        <item id="bp12" image="bp12" />
        <item id="bp13" image="bp13" />
        <item id="bp14" image="bp14" />
```

```
        <item id="bp15" image="bp15" />
    </gallery>
</group>
```

На рис. 22.14 показана первая коллекция, причем названия месяцев отображаются в двух столбцах. Параметр `onAction` выполняет процедуру обратного вызова `MonthSelected`, которая отображает выбранный месяц (эта процедура хранится в виде параметра `Id`).

```
Sub MonthSelected(control As IRibbonControl, _
    id As String, index As Integer)
    MsgBox "You selected " & id
End Sub
```

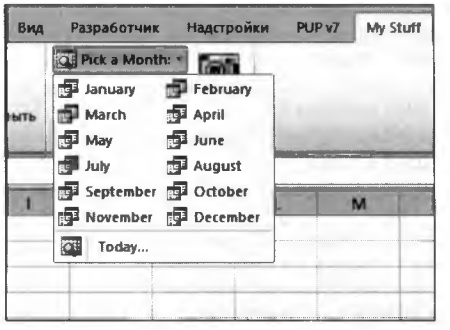


Рис. 22.14. Коллекция отображает названия месяцев и кнопку

Коллекция `Pick a Month` также включает кнопку, с которой связана собственная процедура обратного вызова (под названием `Today`).

```
Sub ShowToday(control As IRibbonControl)
    MsgBox "Today is " & Date
End Sub
```

Вторая коллекция, показанная на рис. 22.15, отображает 15 фотографий.

Фотографии хранятся в файле рабочей книги, в папке `images`, которая находится в папке `customUI`. Операция по добавлению изображений требует наличия папки `_rels`, в которой находится список связей. Для просмотра этого файла добавьте к нему расширение `.zip`, после чего ознакомьтесь с его содержимым.

Пример элемента управления `DynamicMenu`

Один из наиболее интересных элементов управления ленты — элемент управления `DynamicMenu`. Используя этот элемент управления, код VBA передает XML-данные элементу управления. Благодаря этому обеспечивается основа для меню, которые изменяются на основе содержимого.

Настройка элемента управления `DynamicMenu` — непростая задача, однако именно этот элемент управления обеспечивает наибольшую гибкость, позволяя средствами VBA изменять ленту в динамическом режиме.

Я создал простой пример, демонстрирующий применение элемента управления `DynamicMenu`. Этот пример отображает различные меню для каждого из трех листов, находящихся в рабочей книге. На рис. 22.16 показано меню, которое появляется при акти-

визации листа Лист1. При активизации листа процедура VBA отсылает XML-код, заданный для этого листа. В рассматриваемом примере XML-код хранится в рабочих листах, что облегчает его чтение. Также XML-разметка может сохраняться в коде в виде строковой переменной.

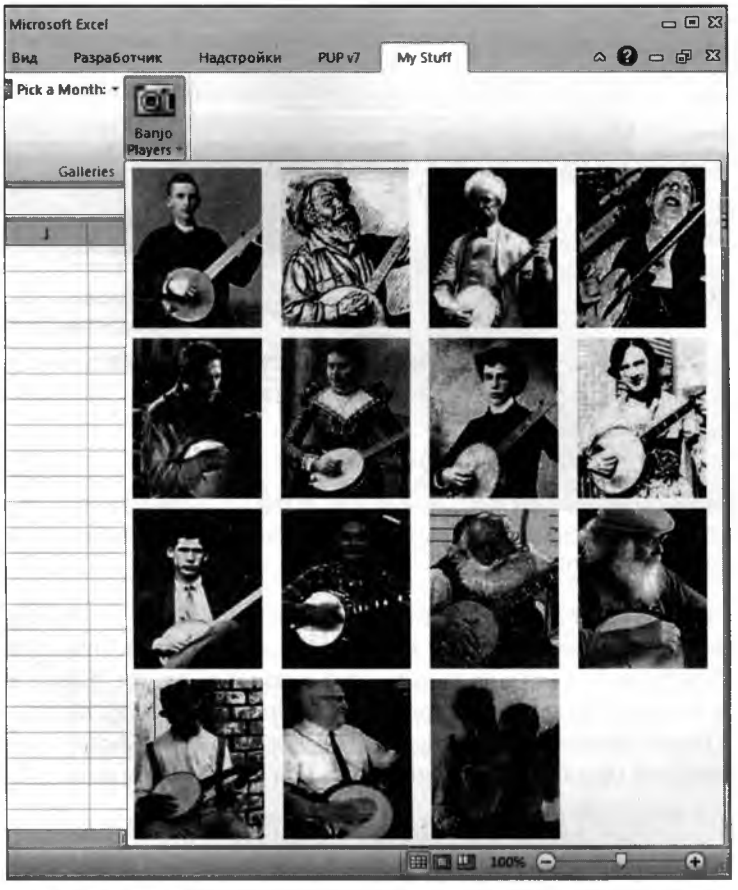


Рис. 22.15. Коллекция фотографий

Ниже приводится код RibbonX, который создает новую вкладку, новую группу, а также новый элемент управления DynamicMenu.

```
<customUI xmlns="http://schemas.microsoft.com/ _
   office/2006/01/customui"
   onLoad="ribbonLoaded">
   <ribbon>
      <tabs>
         <tab id="CustomTab" label="Dynamic">
            <group id="group1" label="Dynamic Menu Demo">
               <dynamicMenu id="DynamicMenu"
                  getContent="dynamicMenuContent"
                  imageMso="RegionLayoutMenu"
                  size = "large"
                  label="Sheet-Specific Menu"/>
            </group>
```

```
    </tab>
    </tabs>
    </ribbon>
</customUI>
```

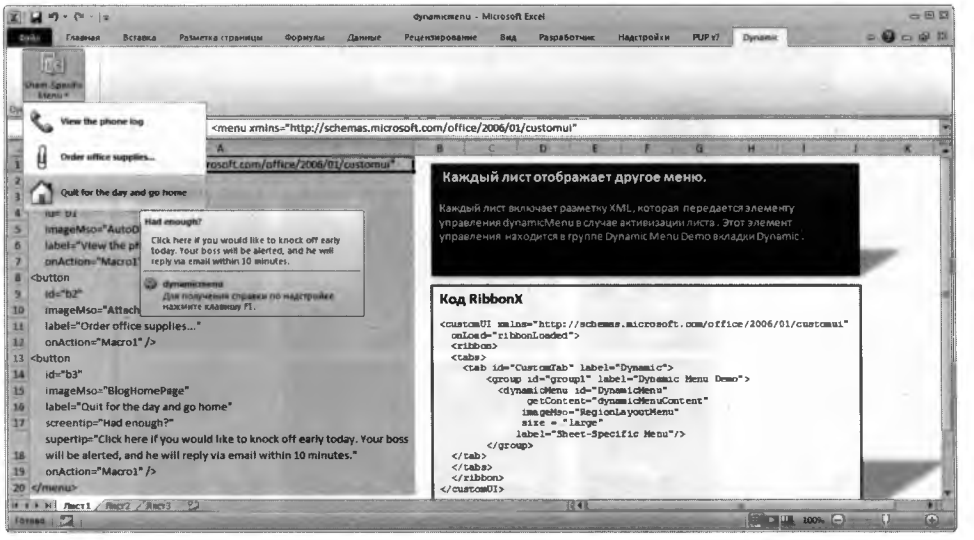


Рис. 22.16. С помощью элемента управления `DynamicMenu` можно создать меню, которое изменяется в зависимости от контекста

В этом примере требуется определить способ деактивизации ленты при активизации пользователем нового листа. При этом используется метод, аналогичный примененному для отображения разрывов страниц ранее в этой главе (см. раздел "Еще один пример кода RibbonX"), — тогда объявлялась глобальная переменная `MyRibbon`, имеющая тип `IRibbonUI`. Также применялась процедура `Workbook_SheetActivate`, которая вызывалась процедурой `UpdateDynamicRibbon` при активизации нового листа.

```
Sub UpdateDynamicRibbon()
'   Отмена выделения ленты для вызова процедуры dynamicMenuContent
    On Error Resume Next
    MyRibbon.Invalidate
    If Err.Number <> 0 Then
       MsgBox "Lost the Ribbon object. Save and reload."
    End If
End Sub
```

Процедура `UpdateDynamicRibbon` отменяет выделение объекта `MyRibbon`, что приводит к выполнению процедуры обратного вызова VBA `dynamicMenuContent` (на эту процедуру ссылается параметр `getContent` в коде RibbonX). Обратите внимание на код обработки ошибок. В результате операций по редактированию, выполняемых в коде VBA, уничтожается объект `MyRibbon`, который был создан при открытии рабочей книги. Попытка отмены выделения несуществующего объекта приводит к появлению ошибки, а также к отображению сообщения, в котором говорится о необходимости сохранения и повторного открытия рабочей книги. К сожалению, единственный способ повторного создания объекта `MyRibbon` — повторное открытие рабочей книги.

Ниже приводится код процедуры dynamicMenuContent. Эта процедура выполняет циклический просмотр ячеек в столбце A активного листа, считывает XML-код, а также сохраняет его в переменной XMLcode. Как только XML-код будет добавлен в полном объеме, он передается аргументу returnedVal. В результате элемент управления DynamicMenu получает новый код, в результате чего отображается другой набор параметров меню.

```
Sub dynamicMenuContent(control As IRibbonControl, _
   ByRef returnedVal)
      Dim r As Long
      Dim XMLcode As String
'     Считывание XML-кода с активного листа
      For r = 1 To Application.CountA(Range("A:A"))
         XMLcode = XMLcode & ActiveSheet.Cells(r, 1) & " "
      Next r
      returnedVal = XMLcode
End Sub
```

Компакт-диск

Рабочая книга, содержащая этот пример, находится на прилагаемом компакт-диске в файле dynamicmenu.xlsm.

Некоторые замечания о настройке ленты

А теперь еще несколько замечаний, которые сослужат вам службу в процессе исследования чудесного мира настройки ленты Excel.

- При работе с лентой убедитесь в том, что включен режим отображения ошибок. Дополнительные сведения об этом режиме можно найти во врезке "Просмотр ошибок".
- Помните о том, что код RibbonX чувствителен к изменению регистра символов.
- При именовании идентификаторов элементов управления применяется английский язык, причем они сохраняют свои названия во всех версиях Excel. Поэтому измененная лента будет работоспособной независимо от языка, используемого в вашей версии Excel.
- Результат изменения ленты отображается только в том случае, когда содержащая код RibbonX рабочая книга будет активной. Если нужно, чтобы изменения ленты отображались для каждой рабочей книги, код RibbonX следует оформить в виде надстройки.
- Встроенные элементы управления автоматически масштабируются при изменении размеров окна Excel. Пользовательские элементы управления всегда имеют одни и те же размеры, т.е. не масштабируются.
- Добавление или удаление элементов управления из встроенной группы ленты невозможно.
- Пользователь может скрывать вкладки. Для этого используется следующий код RibbonX (скрываются три вкладки).

```
<customUI xmlns="http://schemas.microsoft.com/office/2006/01/
   customui">
<ribbon>
   <tabs>
      <tab idMso="TabPageLayoutExcel" visible="false" />
      <tab idMso="TabData" visible="false" />
      <tab idMso="TabReview" visible="false" />
```

```
   </tabs>
</ribbon>
</customUI>
```

- Можно также скрыть группы в составе вкладки. Ниже представлен код RibbonX, который скрывает четыре группы на вкладке **Вставка** (Insert), — остается лишь группа **Диаграммы** (Charts).

```
<customUI xmlns="http://schemas.microsoft.com/office/2006/01/
   customui">
<ribbon>
   <tabs>
      <tab idMso="TabInsert">
      <group idMso="GroupInsertTablesExcel" visible="false" />
      <group idMso="GroupInsertIllustrations" visible="false" />
      <group idMso="GroupInsertLinks" visible="false" />
      <group idMso="GroupInsertText" visible="false" />
   </tab>
   </tabs>
</ribbon>
</customUI>
```

- Встроенному элементу управления можно назначить пользовательский макрос. Подобная операция называется изменением назначения элемента управления. Ниже представлен код RibbonX, который перехватывает три встроенные команды.

```
<customUI xmlns="http://schemas.microsoft.com/office/2006/01/
   customui">
<commands>
   <command idMso="FileSave" onAction="mySave"/>
   <command idMso="FilePrint" onAction="myPrint"/>
   <command idMso="FilePrintQuick" onAction="myPrint"/>
</commands>
</customUI>
```

- Можно также создать код RibbonX, который отключает один или несколько встроенных элементов управления. Приведенный ниже пример кода отключает команду **Вставить клип** (Insert ClipArt).

```
<customUI xmlns="http://schemas.microsoft.com/office/2006/01/
   customui">
<commands>
   <command idMso="ClipArtInsert" enabled="false"/>
</commands>
</customUI>
```

- Если в вашем распоряжении две или более рабочих книг (или надстроек), которые добавляют элементы управления в одну и ту же группу ленты, следует гарантировать использование ими одного и того же пространства имен. Для этого проверьте тег `<CustomUI>`, который находится в верхней части кода RibbonX.

Создание "старомодных" панелей инструментов

Если вы полагаете, что настройка ленты несколько обременительна, можете создать простую настраиваемую панель инструментов с помощью объекта `CommandBar`, используемого в предыдущих версиях Excel. Эта методика великолепно подходит для тех рабочих книг, с которыми будете работать только вы. Она также обеспечивает быстрый доступ к макросам.

В этом разделе будет представлен соответствующий пример кода, который вы сможете легко адаптировать в соответствии со своими потребностями. Я не буду обременять вас излишними подробностями. Дополнительные сведения, касающиеся объекта CommandBar, вы сможете найти в Интернете или в предыдущих изданиях этой книги. Объекты CommandBar обладают намного большими возможностями, чем представленный в данном разделе пример.

Ограничения, присущие "старомодным" панелям в Excel 2010

Если вы захотите создать панель инструментов для Excel 2010, помните о следующих ограничениях:

- она не может быть "плавающей";
- она будет отображаться в группе Настраиваемые панели инструментов (Custom Toolbars) вкладки Надстройки (Add-Ins) наряду с другими панелями инструментов;
- некоторые из свойств и методов объекта CommandBar просто игнорируются Excel.

Код панели инструментов

При выполнении кода, рассматриваемого в этом разделе, предполагается, что рабочая книга включает два макроса (Macro1 и Macro2). Также предполагается, что панель инструментов создается при открытии рабочей книги и удаляется — при ее закрытии.

Примечание

В отличие от результатов изменений ленты, настраиваемые панели отображаются независимо от того, активизирована ли рабочая книга.

В модуле кода ThisWorkbook (ЭтаКнига) введите следующую процедуру. Во-первых, вызывается процедура, которая создает панель инструментов при открытии рабочей книги. Во-вторых, вызывается процедура, которая удаляет панель инструментов после закрытия рабочей книги.

```
Private Sub Workbook_Open()
   Call CreateToolbar
End Sub
Private Sub Workbook_BeforeClose(Cancel As Boolean)
   Call DeleteToolbar
End Sub
```

Перекрестная ссылка

В главе 19 описывалась потенциальная проблема, которая связана с событием Workbook_BeforeClose. После запуска на выполнение обработчика событий Workbook_BeforeClose отображается сообщение Excel "Не хотите ли сохранить..." (Do you want to save...). Если щелкнуть на кнопке Отмена (Cancel), рабочая книга остается открытой, но элементы настраиваемого меню будут удалены. В главе 19 также рассматривался способ решения этой проблемы.

Ниже представлен код процедуры CreateToolbar.

```
Const TOOLBARNAME As String = "MyToolbar"
Sub CreateToolbar()
```

```
    Dim TBar As CommandBar
    Dim Btn As CommandBarButton

'   Удаление существующей панели инструментов
    On Error Resume Next
    CommandBars(TOOLBARNAME).Delete
    On Error GoTo 0

'   Создание панели инструментов
    Set TBar = CommandBars.Add
    With Tbar
        .Name = TOOLBARNAME
        .Visible = True
End With

'   Добавление кнопки
    Set Btn = TBar.Controls.Add(Type:=msoControlButton)
    With Btn
        .FaceId = 300
        .OnAction = "Macro1"
        .Caption = "Подсказка для макроса1"
    End With
'   Добавление другой кнопки
    Set Btn = TBar.Controls.Add(Type:=msoControlButton)
    With Btn
        .FaceId = 25
        .OnAction = "Macro2"
        .Caption = "Подсказка для макроса2"
    End With
End Sub
```

Компакт-диск

Рабочая книга, включающая только что рассмотренный код, находится на прилагаемом компакт-диске в файле `old-style toolbar.xlsm`.

На рис. 22.17 показана панель инструментов, включающая две кнопки.

В рассматриваемом случае использовалась константа уровня модуля под названием TOOLBAR, в которой хранится имя панели инструментов. Это имя также используется процедурой `DeleteToolbar`, поэтому применение константы гарантирует, что обе процедуры будут работать с одним и тем же именем.

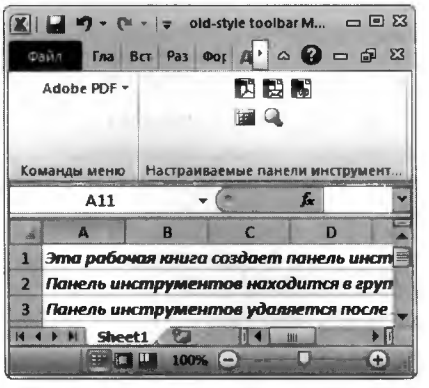


Рис. 22.17. "Старомодная" панель инструментов, находящаяся в группе Настраиваемые панели инструментов (Custom Toolbars) вкладки Надстройки (Add_Ins)

Процедура начинает свою работу с удаления существующей панели инструментов, которая имеет такое же имя (при наличии подобной панели инструментов). Наличие подобного оператора удаления полезно на этапе разработки, а также позволяет устранять ошибки, являющиеся следствием попыток установки панели инструментов, использующей дублирующееся имя.

Созданная панель инструментов использует метод `Add` объекта `CommandBar`. Две кнопки добавляются с помощью метода `Add` объекта `Controls`. Каждая кнопка включает следующие три свойства:

- `FaceID` — число, которое определяет отображаемое на кнопке изображение;
- `OnAction` — макрос, который выполняется после щелчка на кнопке;
- `Caption` — экранная подсказка, отображаемая после установки указателя мыши над кнопкой.

Совет

Вместо установки значения свойства `FaceID`, установите значение свойства `Picture`, используя любое из изображений `imageMso`. Например, следующий оператор отображает зеленую "галочку".

```
.Picture = Application.CommandBars.GetImageMso _
   ("AcceptInvitation", 16, 16)
```

Для получения дополнительных сведений относительно изображений `imageMso` обратитесь к врезке "Использование изображений `imageMso`".

После закрытия рабочей книги вызывается процедура обработки событий `Workbook_BeforeClose`, которая вызывает метод `DeleteToolbar`.

```
Sub DeleteToolbar()
   On Error Resume Next
   CommandBars(TOOLBARNAME).Delete
   On Error GoTo 0
End Sub
```

Глава 23

Работа с контекстными меню

В этой главе...

- Обзор объекта CommandBar
- Настройка контекстных меню с помощью VBA
- Контекстные меню и события

В Excel 2010 роль объекта `CommandBar` намного меньше, чем в предыдущих версиях Excel. В этой главе рассматривается применение объекта `CommandBar` для настройки контекстных меню.

Обзор объекта `CommandBar`

Объект `CommandBar` применяется в следующих трех элементах пользовательского интерфейса Excel:

- настраиваемые панели инструментов;
- настраиваемые меню;
- настраиваемые контекстные меню (вызываются щелчком правой кнопкой мыши).

В Excel 2010 объект `CommandBar` утратил свои былые лидирующие позиции. Если созданный вами код VBA предназначен для настройки меню или панели инструментов, Excel перехватывает этот код, игнорируя при этом многие команды. Как отмечалось в главе 22, настройки меню и панели инструментов, выполненные с помощью объекта `CommandBar`, отображаются в группах **Команды меню** (Menu Commands) и **Настраиваемые панели инструментов** (Custom Toolbars), которые находятся во вкладке **Надстройки** (Add-Ins). Поэтому объект `CommandBar` в Excel 2010 ограничен операциями с контекстными меню.

В этом разделе рассматриваются основные сведения, относящиеся к объекту `CommandBar`.

Типы объектов CommandBar

В Excel поддерживаются три типа объектов CommandBar, которые различаются значениями свойства Type. Это свойство может принимать одно из следующих трех значений:

- msoBarTypeNormal — панель инструментов (Type = 0);
- msoBarTypeMenuBar — панель меню (Type = 1);
- msoBarTypePopUp — контекстное меню (Type = 2).

Несмотря на то что панели инструментов и меню не применяются в Excel 2010, эти элементы пользовательского интерфейса остаются в составе объектной модели (для поддержки совместимости с другими приложениями). Но если в Excel 2010 попытаться отобразить объект CommandBar, для которого значение свойства Type равно 0 или 1, успеха вы не достигнете. А в Excel 2003 следующий оператор отобразит на экране панель инструментов Standard (Стандартная):

```
CommandBars("Standard").Visible = True
```

В Excel 2010 этот оператор игнорируется.

В этой главе рассматриваются объекты CommandBar, для которых значение свойства Type равно 2 (контекстные меню).

Отображение контекстных меню

В Excel 2010 включено 65 контекстных меню. Каким образом можно получить информацию о них? Я запустил процедуру ShowShortcutMenuNames (ее код приведен ниже), которая осуществляет циклический обход всех объектов CommandBar. Если значение свойства Type будет msoBarTypePopUp (встроенная константа, которая имеет значение 2), в рабочем листе отображается индекс и имя объекта CommandBar.

```
Sub ShowShortcutMenuNames()
   Dim Row As Long
   Dim cbar As CommandBar
   Row = 1
   For Each cbar In CommandBars
      If cbar.Type = msoBarTypePopUp Then
         Cells(Row, 1) = cbar.Index
         Cells(Row, 2) = cbar.Name
         Row = Row + 1
      End If
   Next cbar
End Sub
```

На рис. 23.1 показана часть вычислений, являющихся результатом выполнения этой процедуры. Диапазон значений индекса контекстного меню — от 21 до 148. Также обратите внимание на то, что не все имена являются уникальными. Например, объекты CommandBar с индексами 36 и 38 включают Имя ячейки (Name of Cell). Это связано с тем, что щелчок правой кнопкой мыши на ячейке приводит к появлению различных контекстных меню, если в одном из случаев выбран режим Разметка страницы (Page Break Preview).

Компакт-диск

Этот пример находится на прилагаемом к книге компакт-диске в файле show shortcut menu names.xlsm.

Ссылки на объекты CommandBar

Ссылаться на конкретный объект CommandBar можно по его индексу (свойство Index) или имени (свойство Name). Например, следующие выражения ссылаются на контекстное меню, которое отображается после щелчка правой кнопкой мыши на рабочем столе Excel (область, отображаемая при отсутствии открытых документов).

```
Application.CommandBars (44)
Application.CommandBars("Desktop")
```

Коллекция CommandBars входит в состав объекта Application. При установке ссылки на эту коллекцию в обычном модуле VBA или в модуле для листа, можно не указывать ссылку на объект Application. Например, следующий оператор (находящийся в стандартном модуле VBA) отображает имя объекта в коллекции CommandBars, который имеет индекс 44.

```
MsgBox CommandBars(44).Name
```

При установке ссылки на коллекцию CommandBars из модуля кода для объекта ThisWorkbook (ЭтаКнига) предваряйте ее ссылкой на объект Application.

```
MsgBox Application.CommandBars(44).Name
```

Примечание

К сожалению, значения свойства Index не остаются постоянными в различных версиях Excel. Например, в Excel 2007 значения свойства Index для объекта CommandBar, равные 36 и 39, соответствуют имени Name of Cell (Имя ячейки). В Excel 2010 этим двум значениям индекса соответствует название Name of Column (Имя столбца). Поэтому лучше использовать имена, а не значения индекса.

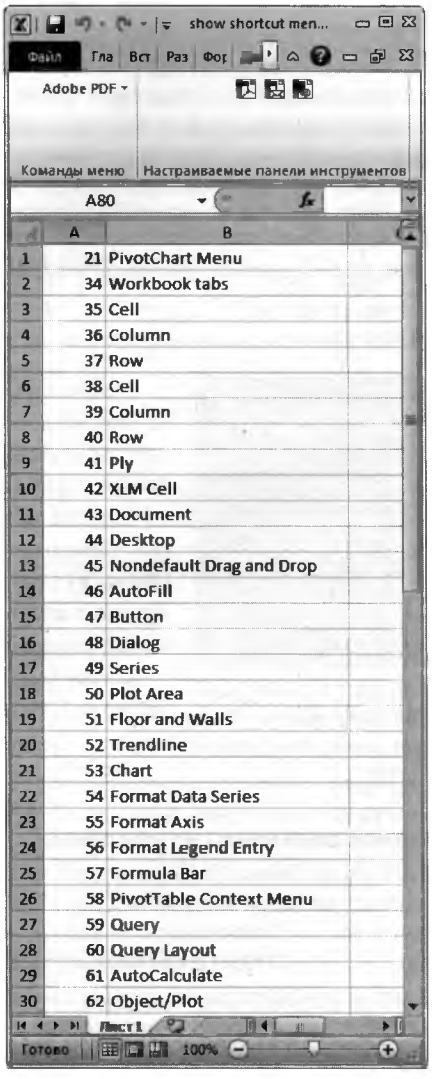


Рис. 23.1. Простой макрос отображает перечень всех контекстных меню

Установка ссылок на элементы управления в объекте CommandBar

Объект CommandBar включает объекты Control, которые являются кнопками или элементами меню. На элемент управления можно ссылаться с помощью свойства Index или Caption. Ниже представлена простая процедура, которая отображает заголовок первого элемента контекстного меню Cell.

```
Sub ShowCaption()
   MsgBox Application.CommandBars("Cell"). _
      Controls(1).Caption
End Sub
```

Следующая процедура отображает значение свойства `Caption` для каждого элемента управления в контекстном меню, которое появляется после щелчка правой кнопкой мыши на ярлычке листа (это контекстное меню называется `Ply`).

```
Sub ShowCaptions()
   Dim txt As String
   Dim ctl As CommandBarControl
   For Each ctl In CommandBars("Ply").Controls
      txt = txt & ctl.Caption & vbNewLine
   Next ctl
   MsgBox txt
End Sub
```

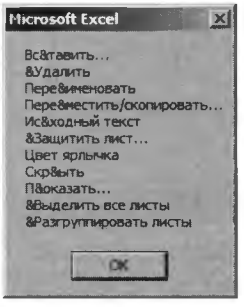


Рис. 23.2. Отображение свойства `Caption` для элементов управления

После выполнения этой процедуры на экране появляется окно сообщения, показанное на рис. 23.2. Знак амперсанда применяется для выделения подчеркнутой буквы в тексте. Эта буква вызывает на выполнение элемент меню.

Иногда объекты `Control` контекстного меню включают другие элементы `Control`. Например, элемент управления `Filter` (Фильтр), находящийся в контекстном меню ячейки, содержит другие элементы управления. В этом случае `Filter` — это подменю, которое включает дополнительные элементы.

Следующий оператор отображает первый элемент подменю `Filter` (Фильтр):

```
MsgBox CommandBars("Cell").Controls("Filter").
   Controls(1).Caption
```

Поиск элемента управления

Если создаваемый вами код рассчитан на применение в различных версиях Excel, не следует использовать свойство `Caption` для доступа к определенному элементу контекстного меню. Свойство `Caption` зависит от используемого языка, поэтому код не будет работать, если его перенести в среду Excel, ориентированную на работу с другим языком.

В этом случае обратитесь к методу `FindControl` и воспользуйтесь идентификатором (`ID`) элемента управления (эти средства не зависят от используемого языка). Предположим, что нужно отключить пункт Переименовать (Rename) контекстного меню, который отображается после щелчка мышью на ярлычке листа. Если с рабочей книгой будут работать исключительно пользователи англоязычной версии Excel, воспользуйтесь следующим оператором:

```
CommandBars("Ply").Controls("Rename").Enabled = False
```

Если нужно обеспечить правильную работу этой команды с неанглийскими версиями Excel, следует знать идентификатор элемента управления. Следующий оператор сообщает нам, что значение идентификатора (`ID`) равно 889:

```
MsgBox CommandBars("Ply").Controls("Rename").ID
```

Затем для отключения этого элемента управления используйте следующий оператор:

```
CommandBars.FindControl(ID:=889).Enabled = False
```

Имена объектов `CommandBar` не изменяются в версиях Excel, отличающихся от английской, поэтому ссылка `CommandBars("Desktop")` будет всегда работоспособной.

Свойства элементов управления CommandBar

Элементы управления CommandBar включают ряд свойств, которые определяют их внешний вид и поведение. Ниже представлен список некоторых чаще всего используемых свойств объекта CommandBar.

- Caption. Этот текст отображается на элементе управления. Если элемент управления отображает только изображение, значение свойства Caption отображается после установки указателя мыши над элементом управления.
- ID. Уникальный числовой идентификатор элемента управления.
- FaceID. Это число представляет изображение, отображаемое рядом с текстом, заданным для элемента управления.
- Type. Это значение определяет, является ли элемент управления кнопкой (msoControlButton) или подменю (msoControlPopup).
- Picture. Изображение, отображаемое рядом с текстом, заданным для элемента управления.
- BeginGroup. Значение этого свойства равно True, если перед элементом управления отображается разделитель.
- OnAction. Название макроса VBA, вызываемого на выполнение после щелчка мышью на элементе управления.
- BuiltIn. Значение этого свойства равно True, если относится к встроенному элементу управления Excel.
- Enabled. Значение этого свойства равно True, если на элементе управления произведен щелчок мышью.
- Visible. Значение этого свойства равно True, если элемент управления является видимым. Многие из контекстных меню содержат скрытые элементы управления.
- ToolTipText. Текст, отображаемый после установки указателя мыши над элементом управления. (Не применяется для контекстных меню.)

Отображение всех элементов контекстного меню

Процедура ShowShortcutMenuItems, код которой приведен ниже, создает таблицу, в которой находятся все элементы управления первого уровня, которые относятся к каждому элементу контекстного меню. Для каждого элемента управления отображаются значения свойств Index, Name, ID, Caption, Type, Enabled и Visible.

```
Sub ShowShortcutMenuItems()
  Dim Row As Long
  Dim Cbar As CommandBar
  Dim ctl As CommandBarControl
  Range("A1:G1") = Array("Index", "Name", "ID", "Caption", _
     "Type", "Enabled", "Visible")
  Row = 2
  Application.ScreenUpdating = False
  For Each Cbar In Application.CommandBars
     If Cbar.Type = 2 Then
        For Each ctl In Cbar.Controls
           Cells(Row, 1) = Cbar.Index
```

```
            Cells(Row, 2) = Cbar.Name
            Cells(Row, 3) = ctl.ID
            Cells(Row, 4) = ctl.Caption
            If ctl.Type = 1 Then
               Cells(Row, 5) = "Кнопка"
            Else
               Cells(Row, 5) = "Подменю"
            End If
            Cells(Row, 6) = ctl.Enabled
            Cells(Row, 7) = ctl.Visible
            Row = Row + 1
         Next ctl
      End If
   Next Cbar
End Sub
```

На рис. 23.3 показана часть выводимого результата.

	A	B	C	D	E	F	G
16	34	Workbook tabs	957	Сп&исок листов	Кнопка	ИСТИНА	ЛОЖЬ
17	34	Workbook tabs	957	Сп&исок листов	Кнопка	ИСТИНА	ЛОЖЬ
18	34	Workbook tabs	957	Сп&исок листов	Кнопка	ИСТИНА	ЛОЖЬ
19	34	Workbook tabs	957	Сп&исок листов	Кнопка	ИСТИНА	ЛОЖЬ
20	34	Workbook tabs	957	Сп&исок листов	Кнопка	ИСТИНА	ЛОЖЬ
21	34	Workbook tabs	957	Сп&исок листов	Кнопка	ИСТИНА	ЛОЖЬ
22	34	Workbook tabs	957	Сп&исок листов	Кнопка	ИСТИНА	ЛОЖЬ
23	34	Workbook tabs	957	Сп&исок листов	Кнопка	ИСТИНА	ЛОЖЬ
24	35	Cell	21	В&ырезать	Кнопка	ИСТИНА	ИСТИНА
25	35	Cell	19	&Копировать	Кнопка	ИСТИНА	ИСТИНА
26	35	Cell	22	Вставит&ь	Кнопка	ИСТИНА	ИСТИНА
27	35	Cell	21437	&Специальная вставка...	Кнопка	ИСТИНА	ИСТИНА
28	35	Cell	3624	Вставить &таблицу	Кнопка	ИСТИНА	ИСТИНА
29	35	Cell	3181	Вс&тавить...	Кнопка	ИСТИНА	ИСТИНА
30	35	Cell	292	&Удалить...	Кнопка	ИСТИНА	ИСТИНА
31	35	Cell	3125	Очистить содер&жимое	Кнопка	ИСТИНА	ИСТИНА
32	35	Cell	31623	&Специальны	Подменю	ИСТИНА	ЛОЖЬ
33	35	Cell	31402	&Фильтр	Подменю	ИСТИНА	ИСТИНА
34	35	Cell	31435	&Сортировка	Подменю	ИСТИНА	ИСТИНА
35	35	Cell	2031	Вставить приме&чание	Кнопка	ИСТИНА	ИСТИНА
36	35	Cell	1592	Удалить при&мечание	Кнопка	ЛОЖЬ	ЛОЖЬ
37	35	Cell	1593	П&оказать или скрыть примечания	Кнопка	ЛОЖЬ	ЛОЖЬ
38	35	Cell	855	Формат &ячеек...	Кнопка	ИСТИНА	ИСТИНА
39	35	Cell	1966	Выбрать из раскрывающегося списка...	Кнопка	ИСТИНА	ИСТИНА
40	35	Cell	1614	&Показать фонетическое поле	Кнопка	ИСТИНА	ЛОЖЬ
41	35	Cell	13380	Присвоить&ь имя...	Кнопка	ИСТИНА	ИСТИНА
42	35	Cell	1576	Гиперссы&лка...	Кнопка	ИСТИНА	ИСТИНА
43	35	Cell	1577	Измени&ть гиперссылку...	Кнопка	ЛОЖЬ	ЛОЖЬ
44	35	Cell	1015	&Открыть гиперссылку	Кнопка	ЛОЖЬ	ЛОЖЬ
45	35	Cell	3626	&Удалить гиперссылку	Кнопка	ЛОЖЬ	ЛОЖЬ

Рис. 23.3. Отображение всех элементов контекстного меню

После запуска на выполнение макроса `ShowShortcutMenuItems` вы обнаружите, что многие контекстные меню содержат скрытые или отключенные элементы управления. Эти пункты меню представляют элементы управления, которые недоступны для текущего контекста. Например, контекстное меню **Рабочий стол** (Desktop) с индексом 44 включает следующие элементы меню:

- &Создать... (&New...);
- &Открыть... (&Open...);
- Сохранить &рабочую область... (Save &Workspace...);
- &Вычислить (&Calculate Now);
- П&олноэкранный режим (F&ull Screen).

Пункт меню П&олноэкранный режим (Full Screen) обычно скрыт — до тех пор, пока Excel не переходит в полноэкранный режим. Если же последнее имеет место, этот пункт меню становится видимым, а его заголовок изменяется на &Выйти из полноэкранного режима (&Close Full Screen).

Компакт-диск

Рассматриваемая в этом разделе рабочая книга находится на прилагаемом компакт-диске в файле `show shortcut menu items.xlsm`.

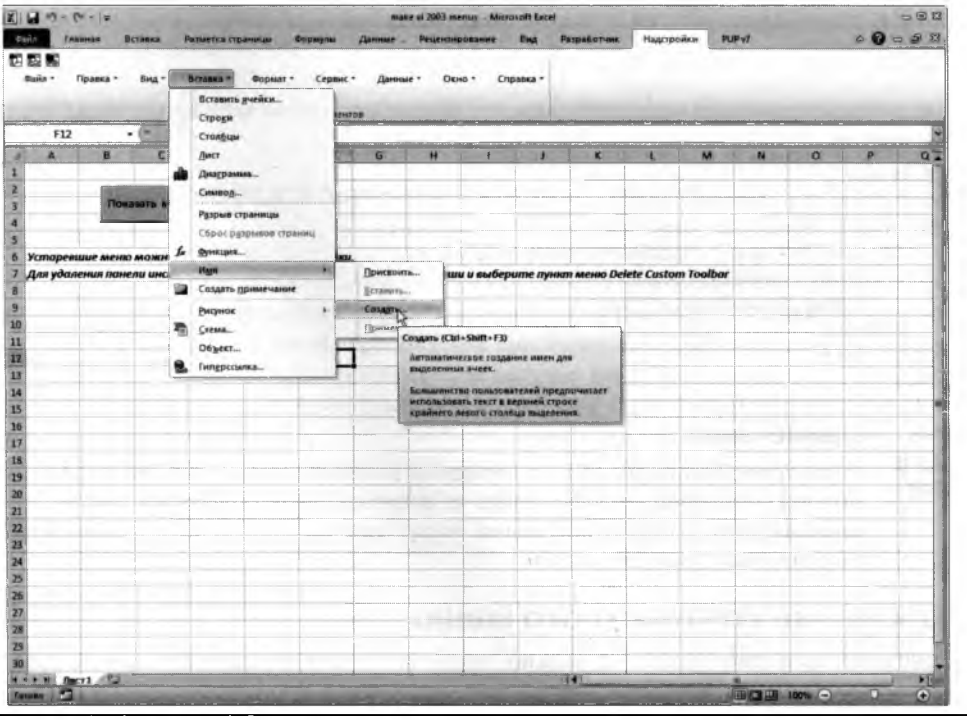
Отображение меню Excel 2003

Одно из встроенных контекстных меню называется `Built-In Menus`. Это меню включает элементы из Excel 2003 (последняя версия Excel, в которой отсутствовала лента). Это контекстное меню не связано с каким-то определенным объектом, а для его отображения можно воспользоваться следующей командой VBA:

```
Application.CommandBars("Built-in Menus").ShowPopup
```

На прилагаемом к книге компакт-диске можно найти пример (`make xl 2003 menus.xlsm`), который содержит код, копирующий контекстные меню на панель инструментов. Панель инструментов отображается на ленте после выбора вкладки Надстройки (Add-Ins). В результате у вас появится возможность использовать меню Excel 2003 при работе в Excel 2010.

Ниже показан внешний вид этих меню.



Настройка контекстных меню с помощью VBA

В этом разделе будут рассмотрены некоторые практические примеры кода VBA, который работает с контекстными меню Excel. Эти примеры дают представление о том, что можно делать с контекстными меню с помощью VBA, а соответствующий им код можно изменять произвольным образом.

Сброс контекстных меню

Метод `Reset` восстанавливает исходное состояние контекстного меню. Следующая процедура возвращает исходное состояние контекстного меню ячейки.

```
Sub ResetCellMenu()
   CommandBars("Cell").Reset
End Sub
```

Как отмечалось ранее, Excel включает два контекстных меню ячейки. Приведенный выше пример кода восстанавливает только первое меню (с индексом 35). Для восстановления исходного состояния второго контекстного меню ячейки подставьте значение его индекса (38) вместо значения индекса первого меню. Поскольку номера индексов отличаются в различных версиях Excel, ниже представлена усовершенствованная процедура, которая восстанавливает оба контекстных меню ячейки.

```
Sub ResetCellMenu()
   Dim cbar As CommandBar
   For Each cbar In Application.CommandBars
      If cbar.Name = "Cell" Then cbar.Enabled = False
   Next cbar
End Sub
```

Следующая процедура восстанавливает исходное состояние всех встроенных панелей инструментов.

```
Sub ResetAll()
   Dim cbar As CommandBar
   For Each cbar In Application.CommandBars
      If cbar.Type = msoBarTypePopup Then
         cbar.Reset
         cbar.Enabled = True
      End If
   Next cbar
End Sub
```

Примечание

Если приложение добавляет элементы в контекстное меню, после закрытия приложения следует их удалить в индивидуальном порядке. Если просто восстановить контекстное меню, будут удалены все настройки, выполненные другими приложениями.

Отключение контекстного меню

Используя свойство `Enabled`, можно отключить все контекстное меню. Если присвоить этому свойству значение `False`, щелчок правой кнопкой мыши на ячейке не приведет к отображению контекстного меню. Следующий оператор отключает контекстное меню ячейки:

```
Application.CommandBars("Cell").Enabled = False
```

Для повторной активизации контекстного меню присвойте свойству `Enabled` значение `True`.

Если нужно отключить *все* контекстные меню, используйте следующую процедуру.

```
Sub DisableAllShortcutMenus()
   Dim cb As CommandBar
   For Each cb In CommandBars
      If cb.Type = msoBarTypePopup Then _
         cb.Enabled = False
   Next cb
End Sub
```

Предупреждение

Состояние отключения контекстных меню "замораживается" между рабочими сеансами. Поэтому перед закрытием Excel лучше восстановить контекстные меню. Для этого измените предыдущую процедуру таким образом, чтобы свойству `Enabled` присваивалось значение `True`.

Отключение элементов контекстного меню

В процессе выполнения приложения иногда нужно отключать один или более элементов определенного контекстного меню. Если элемент отключен, соответствующий ему текст становится светло-серым и щелчок на нем мышью не дает никакого эффекта. Следующая процедура отключает элемент Скрыть (Hide) контекстных меню Строка (Row) и Столбец (Column).

```
Sub DisableHideMenuItems()
   CommandBars("Column").Controls("Скрыть").Enabled = False
   CommandBars("Row").Controls("Скрыть").Enabled = False
End Sub
```

Добавление нового элемента в контекстное меню ячейки

Процедура `AddToShortcut`, код которой приведен ниже, добавляет новый элемент в контекстное меню ячейки. Этот элемент называется Перенос по словам (Toggle Word Wrap). Как вы помните, в Excel имеются два контекстных меню ячейки. Рассматриваемая в этом разделе процедура изменяет стандартное контекстное меню, причем новое меню не совпадает с тем, которое появляется в режиме просмотра Разметка страницы (Page Break Preview).

```
Sub AddToShortCut()
' Добавление элемента меню в контекстное меню ячейки
   Dim Bar As CommandBar
   Dim NewControl As CommandBarButton
   DeleteFromShortcut
   Set Bar = CommandBars("Cell")
   Set NewControl = Bar.Controls.Add _
      (Type:=msoControlButton, _
      temporary:=True)
   With NewControl
      .Caption = "&Перенос по словам"
      .OnAction = "ToggleWordWrap"
      .Picture = Application.CommandBars.GetImageMso _
         ("WrapText", 16, 16)
```

```
        .Style = msoButtonIconAndCaption
    End With
End Sub
```

На рис. 23.4 показан новый элемент меню, который отображается после щелчка правой кнопкой мыши на ячейке. Первая инструкция кода, находящаяся после объявлений переменных, вызывает на выполнение процедуру `DeleteFromShortcut` (код этой процедуры приведен далее в данном разделе). Эта инструкция определяет отображение в контекстном меню ячейки только элемента Перенос по словам (Toggle Word Wrap).

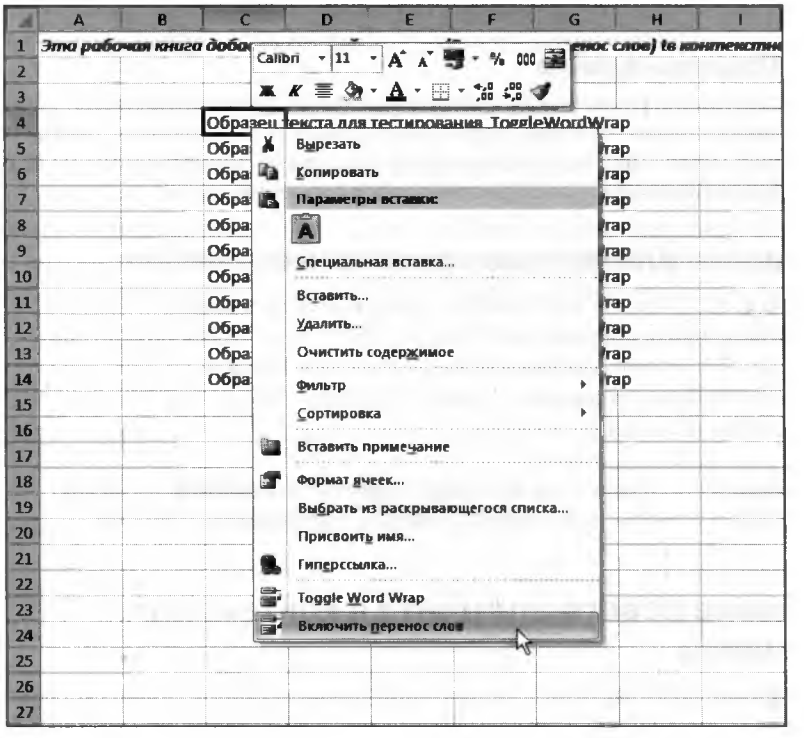


Рис. 23.4. Контекстное меню ячейки, включающее новый элемент

Значение свойства `Picture` устанавливается путем установки ссылки на изображение, используемое для ленты (для команды Перенос по словам (Toggle Word Wrap)).

Для получения дополнительных сведений об изображениях, применяемых для команд ленты, обратитесь к главе 22.

Макрос, вызываемый на выполнение после выбора элемента меню, определяется с помощью свойства `OnAction`. В рассматриваемом случае вызывается макрос `ToggleWordWrap`.

```
Sub ToggleWordWrap()
    CommandBars.ExecuteMso ("WrapText")
End Sub
```

Эта процедура просто вызывает команду ленты Перенос текста (WrapText).

Примечание

Запомните, что изменения контекстного меню оказывают действие до тех пор, пока не будет перезапущена Excel. Другими словами, измененные контекстные меню не восстанавливаются в исходное состояние после закрытия рабочей книги с кодом VBA. Поэтому, если был создан код, изменяющий контекстное меню, предусмотрите процедуру, которая отменяет результаты изменений.

Процедура `DeleteFromShortcut` удаляет новый элемент из контекстного меню ячейки.

```
Sub DeleteFromShortcut()
   On Error Resume Next
   CommandBars("Cell").Controls _
      ("&Перенос по словам").Delete
End Sub
```

Как правило, добавление и удаление элементов контекстного меню производится автоматически. Элемент меню добавляется при открытии рабочей книги и удаляется при ее закрытии. Для реализации этой идеи на практике просто добавьте следующие две процедуры обработки событий в модуль кода `ThisWorkbook` (ЭтаКнига).

```
Private Sub Workbook_Open()
   Call AddToShortCut
End Sub
Private Sub Workbook_BeforeClose(Cancel As Boolean)
   Call DeleteFromShortcut
End Sub
```

Процедура `Workbook_Open` вызывается после открытия рабочей книги, а процедура `Workbook_BeforeClose` — перед ее закрытием. Именно то, что нужно.

Примечание

Элементы, добавленные в контекстное меню, будут доступны для всех рабочих книг, а не только в книге, в которой они были созданы.

Компакт-диск

Рассматриваемая в этом разделе рабочая книга находится на прилагаемом компакт-диске в файле `add to cell shortcut.xlsm`.

Добавление подменю в контекстное меню

Пример, рассматриваемый в этом разделе, добавляет в контекстное меню подменю с тремя опциями. Фактически, подменю добавляется в шесть контекстных меню. На рис. 23.5 показан вид рабочего листа после щелчка правой кнопкой мыши на строке. Каждая из опций подменю вызывает на выполнение макрос, который изменяет регистр символов в выделенных ячейках.

Следующий код создает подменю, а также входящие в его состав опции.

```
Sub AddSubmenu()
' Добавляет подменю к шести контекстным меню
   Dim Bar As CommandBar
   Dim NewMenu As CommandBarControl
   Dim NewSubmenu As CommandBarButton
   DeleteSubmenu
```

```
    Set Bar = CommandBars("Cell")
'   Добавление подменю
    Set NewMenu = Bar.Controls.Add _
        (Type:=msoControlPopup, _
        temporary:=True)
    NewMenu.Caption = "Из&менить регистр"
    NewMenu.BeginGroup = True
'   Добавление первого элемента подменю
    Set NewSubmenu = NewMenu.Controls.Add _
        (Type:=msoControlButton)
    With NewSubmenu
        .FaceId = 38
        .Caption = "&Верхний регистр"
        .OnAction = "MakeUpperCase"
    End With
'   Добавление второго элемента подменю
    Set NewSubmenu = NewMenu.Controls.Add _
        (Type:=msoControlButton)
    With NewSubmenu
        .FaceId = 40
        .Caption = "&Нижний регистр"
        .OnAction = "MakeLowerCase"
    End With
'   Добавление третьего элемента подменю
    Set NewSubmenu = NewMenu.Controls.Add _
        (Type:=msoControlButton)
    With NewSubmenu
        .FaceId = 476
        .Caption = "&Подходящий регистр"
        .OnAction = "MakeProperCase"
    End With
End Sub
```

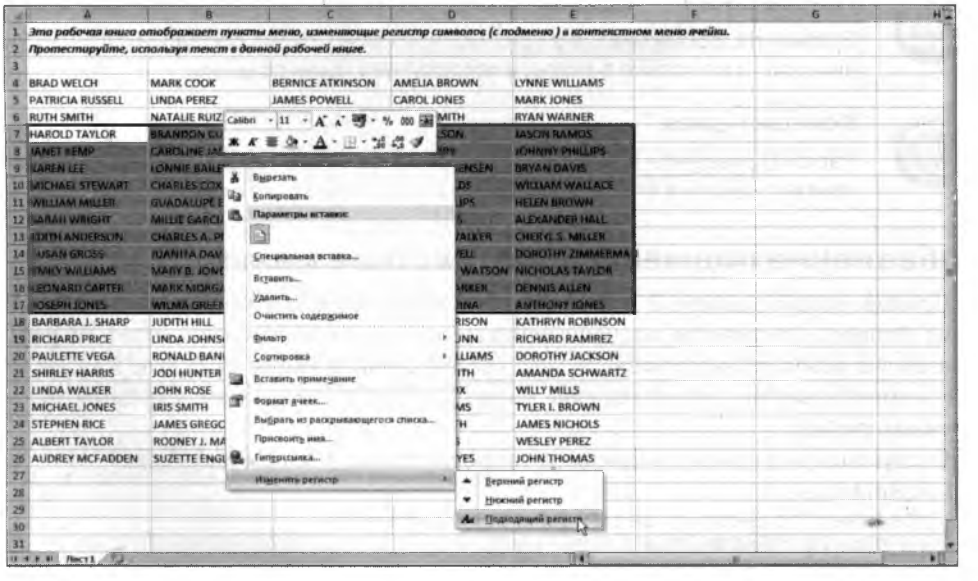


Рис. 23.5. Это контекстное меню включает подменю с тремя опциями

Сначала добавляется подменю, свойство Type которого имеет значение msoControlPopup. Затем добавляются три опции подменю — каждая из них имеет различное свойство OnAction.

Код, предназначенный для удаления подменю, весьма прост.

```
Sub DeleteSubmenu()
   On Error Resume Next
   CommandBars("Cell").Controls("Из&менить регистр").Delete
End Sub
```

Компакт-диск

Описанная в этом разделе рабочая книга находится на прилагаемом компакт-диске в файле shortcut with submenu.xlsm.

Поиск изображений, заданных свойством FaceID

Значок, связанный с элементом контекстного меню, определяется одним из двух свойств.

- **Picture.** Это свойство обеспечивает использование изображения imageMso, заданного для ленты. Пример можно найти в разделе "Добавление нового элемента в контекстное меню ячейки".
- **FaceID.** Наиболее простое в применении свойство FaceID является числовым значением, которое представляет одно из нескольких сотен изображений.

Каким же образом можно найти значение числа, соответствующее конкретному изображению, заданному свойством FaceID? В Excel решения этой проблемы не существует, поэтому я разработал приложение, в окне которого достаточно ввести начальное и конечное значения FaceID. После этого нужно щелкнуть на кнопке, и требуемое изображение появляется на рабочем листе. Каждому изображению присвоено название, которое соответствует значению FaceID. Обратите внимание на следующий рисунок, где показаны значения свойства FaceID, варьирующиеся от 1 до 500. Эта рабочая книга находится на прилагаемом компакт-диске в файле show faceids.xlsm.

Контекстные меню и события

Примеры, рассматриваемые в этом разделе, демонстрируют различные методики программирования контекстных меню, применяемые совместно с событиями.

Перекрестная ссылка

Подробно процесс программирования событий рассмотрен в главе 19.

Автоматическое добавление и удаление меню

Если нужно изменить контекстное меню, которое появляется в момент открытия рабочей книги, используется событие Workbook_Open. Ниже представлен код, который находится в модуле кода для объекта ThisWorkbook (ЭтаКнига) и вызывает процедуру ModifyShortcut (код этой процедуры здесь не показан).

```
Private Sub Workbook_Open()
   Call ModifyShortcut
End Sub
```

Для возврата исходного состояния контекстного меню (до выполнения изменения) воспользуйтесь приведенной ниже процедурой. Эта процедура выполняется перед закрытием рабочей книги и вызывает процедуру RestoreShortcut (код этой процедуры здесь не приводится).

```
Private Sub Workbook_BeforeClose(Cancel As Boolean)
   Call RestoreShortcut
End Sub
```

Если рабочая книга не была сохранена и пользователь пытается закрыть ее, возникает проблема. Программа Excel отображает сообщение "Сохранить изменения?" ("Do you want to save the changes?") после запуска на выполнение обработчика событий Workbook_BeforeClose. Если щелкнуть на кнопке Отмена (Cancel), рабочая книга останется открытой, но ваше настраиваемое меню будет безвозвратно утеряно!

Одно из решений этой проблемы заключается в том, чтобы обойти отображение запроса Excel, создав собственный код (в составе процедуры Workbook_BeforeClose), который отображает запрос о сохранении рабочей книги. Этот код показан ниже.

```
Private Sub Workbook_BeforeClose(Cancel As Boolean)
   If Not Me.Saved Then
   Msg = "Хотите сохранить изменения в "
   Msg = Msg & Me.Name & "?"
   Ans = MsgBox(Msg, vbQuestion + vbYesNoCancel)
   Select Case Ans
      Case vbYes
         Me.Save
      Case vbNo
         Me.Saved = True
      Case vbCancel
         Cancel = True
         Exit Sub
      End Select
   End If
   Call RestoreShortcut
End Sub
```

Эта процедура определяет, была ли сохранена рабочая книга. Если рабочая книга была сохранена, проблем не возникает. Вызывается процедура RestoreShortcut, и рабочая книга закрывается. Но если рабочая книга не сохранялась, отображается окно сообщения, которое дублирует одно из окон, обычно отображаемых Excel. Как только пользователь щелкнет на кнопке Да (Yes), рабочая книга сохраняется, меню удаляется и рабочая книга закрывается. Если щелкнуть на кнопке Нет (No), свойству Saved объекта Workbook присваивается значение True (файл фактически не сохраняется), а меню удаляется. Если щелкнуть на кнопке Отмена (Cancel), событие BeforeClose отменяется, а процедура завершает свою работу без восстановления контекстного меню.

Отключение или сокрытие элементов контекстного меню

Если элемент меню отключен, сопровождающий его текст выделен светло-серым цветом, а щелчок мышью на таком элементе не приведет к какому-либо результату. Если элемент меню скрыт, он не отображается в контекстном меню. Можно также создать код VBA, который активизирует или отключает элементы контекстного меню. Можно также создать код, который скрывает элементы контекстного меню. Основная проблема заключается в корректной обработке событий.

Например, следующий код отключает элемент контекстного меню **Изменить регистр** (Change Case). Этот элемент добавляется в контекстное меню ячейки при активизации листа `Лист2`. Соответствующая процедура находится в модуле кода листа `Лист2`.

```
Private Sub Worksheet_Activate()
   CommandBars("Cell").Controls( _
      "Изменить регистр").Enabled = False
End Sub
```

Для активизации элемента меню при выделении листа `Лист2` добавьте следующую процедуру. В результате применения этой процедуры элемент меню **Изменить регистр** (Change Case) будет доступен все время, за исключением активизации листа `Лист2`.

```
Private Sub Worksheet_Deactivate()
   CommandBars("Cell").Controls( _
      "Изменить регистр").Enabled = True
End Sub
```

Чтобы скрыть элемент меню, используйте свойство `Visible` (вместо свойства `Enabled`).

Создание нового контекстного меню

Можно создать полностью новое контекстное меню, которое будет отображаться в ответ на определенное событие. Представленный ниже код создает контекстное меню `MyShortcut`, которое включает шесть опций. Элементы этого меню обладают собственными свойствами `OnAction`, значения которых настроены таким образом, что вызывается простая процедура, отображающая одну из вкладок диалогового окна **Формат ячеек** (Format Cells), как показано на рис. 23.6.

```
Sub CreateShortcut()
   Set myBar = CommandBars.Add _
      (Name:="MyShortcut", Position:=msoBarPopup, _
      Temporary:=True)

'  Добавление элемента меню
   Set myItem = myBar.Controls.Add(Type:=msoControlButton)
   With myItem
      .Caption = "&Числовой формат..."
      .OnAction = "ShowFormatNumber"
      .FaceId = 1554
   End With

'  Добавление элемента меню
   Set myItem = myBar.Controls.Add(Type:=msoControlButton)
   With myItem
      .Caption = "&Выравнивание..."
```

```
        .OnAction = "ShowFormatAlignment"
        .FaceId = 217
    End With

'   Добавление элемента меню
    Set myItem = myBar.Controls.Add(Type:=msoControlButton)
    With myItem
        .Caption = "&Шрифт..."
        .OnAction = "ShowFormatFont"
        .FaceId = 291
    End With

'   Добавление элемента меню
    Set myItem = myBar.Controls.Add(Type:=msoControlButton)
    With myItem
        .Caption = "&Границы..."
        .OnAction = "ShowFormatBorder"
        .FaceId = 149
        .BeginGroup = True
    End With

'   Добавление элемента меню
    Set myItem = myBar.Controls.Add(Type:=msoControlButton)
    With myItem
        .Caption = "&Узоры..."
        .OnAction = "ShowFormatPatterns"
        .FaceId = 1550
    End With

'   Добавление элемента меню
    Set myItem = myBar.Controls.Add(Type:=msoControlButton)
    With myItem
        .Caption = "&Защита..."
        .OnAction = "ShowFormatProtection"
        .FaceId = 2654
    End With
End Sub
```

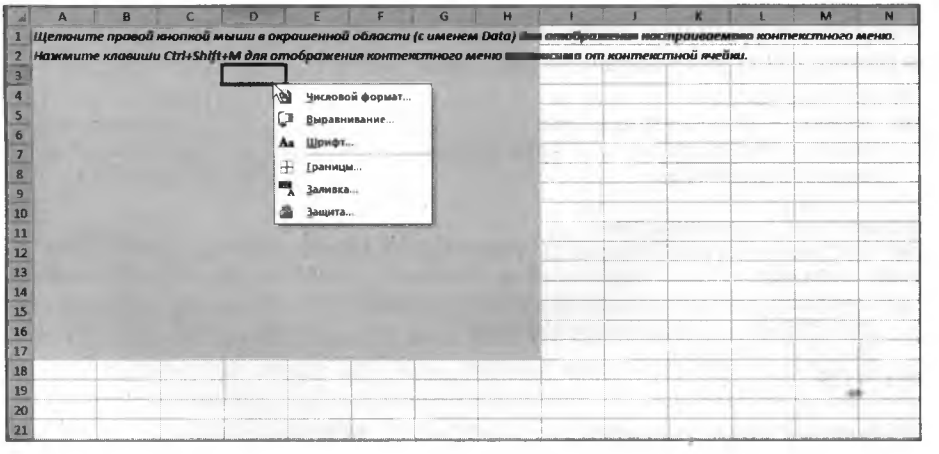


Рис. 23.6. Новое контекстное меню отображается только после щелчка правой кнопкой мыши на окрашенной области рабочего листа

После завершения создания контекстного меню можно его отобразить с помощью метода ShowPopup. Следующая процедура, находящаяся в модуле кода для объекта Worksheet (Лист), вызывается после щелчка на ячейке правой кнопкой мыши.

```
Private Sub Worksheet_BeforeRightClick _
   (ByVal Target As Excel.Range, Cancel As Boolean)
   If Union(Target.Range("A1"), Range("data")).Address = _
      Range("data").Address Then
      CommandBars("MyShortcut").ShowPopup
      Cancel = True
   End If
End Sub
```

Если активная ячейка находится в диапазоне data и пользователь щелкает правой кнопкой мыши, отображается меню MyShortcut. Если аргументу Cancel присваивается значение True, стандартное контекстное меню не отображается. Обратите внимание: мини-панель также не отображается.

Это контекстное меню можно также отобразить без помощи мыши. Создайте простую процедуру, а затем назначьте клавиши быстрого доступа с помощью поля Сочетание клавиш, которое находится в диалоговом окне Макрос (Macro).

```
Sub ShowMyShortcutMenu()
'  Сочетание клавиш <Ctrl+Shift+M>
   CommandBars("MyShortcut").ShowPopup
End Sub
```

Компакт-диск

На прилагаемом к книге компакт-диске находится пример (context-sensitive shortcut menu.xlsm), в котором создается новое контекстное меню, отображаемое вместо стандартного контекстного меню ячейки.

Глава 24

Предоставление справки в приложениях

В этой главе...

- Справка в приложениях Excel
- Справочная система, созданная с помощью компонентов Excel
- Отображение справки в окне браузера
- Использование средства HTML Help
- Связывание файлов справочного руководства с приложением

В этой главе рассмотрены способы предоставления справки пользователям Excel.

Справка в приложениях Excel

Если в Excel разрабатывается довольно сложное приложение, то может возникнуть необходимость в создании справочного руководства для потенциальных пользователей приложения. Таким образом, пользователям будет комфортно работать с приложением, и они не станут беспокоить вас телефонными звонками. Еще одним преимуществом интерактивного справочного руководства является постоянная его доступность (это руководство невозможно потерять или "похоронить" под огромной стопкой книг).

Справочное руководство для пользователей можно добавить к приложению несколькими способами, различающимися по степени сложности. Метод, который будет вами выбран, зависит от масштаба и сложности конкретного приложения, а также от того, сколько времени вы можете себе позволить потратить на создание справочного руководства для приложения. Некоторые приложения нуждаются исключительно в простом наборе инструкций по запуску и управлению. Для других приложений необходимо создавать полноценную справочную систему, обеспечивающую возможности полноценного поиска. Но для большинства приложений требуется справочное руководство среднего уровня — достаточно сложное для создания без подготовки, но простое для изучения.

Предметом рассмотрения настоящей главы являются пользовательские справочные системы одной из следующих категорий.

- **Неофициальная справочная система.** Это справка, которая используется стандартными компонентами Excel (например, диалоговыми окнами `UserForm`).
- **Официальная справочная система.** Эта справочная система использует компилированный CHM-файл, созданный с помощью Microsoft HTML Help Workshop.

Диалоговая система

В последнее время каждая программа оснащается тем, что мы называем *диалоговой справочной системой*. Как и следовало ожидать, под этим понимается справочная информация, размещаемая в Интернете. Но некоторые пользователи до сих пор заблуждаются, полагая, что все справочные данные по использованию программы хранятся на локальном жестком диске.

Поэтому термин "справочная система" обозначает информацию, предоставленную приложением (т.е. сохраненную на жестком диске). Начиная с версии Excel 2003 все изменилось. Справочная система в Excel 2003 и более поздних версиях является интерактивной в полном смысле этого слова. Пользователь может просматривать ранее сохраненную на локальном диске справочную информацию либо (через подключение к Интернету) осуществлять поиск требуемых сведений на веб-сайте Microsoft.

Создание компилированного файла справки — непростая задача. К этому методу следует прибегать в том случае, когда создается сложное приложение или вашим приложением будет пользоваться много людей.

Примечание

Начиная с версии Microsoft Office 2007 компания Microsoft отказалась от применения файлов справки в формате `CHM` в своих приложениях Office. Взамен начала применяться иная (и намного более сложная) справочная система под названием MS Help 2. Эта система не рассматривается в данной книге.

Компакт-диск

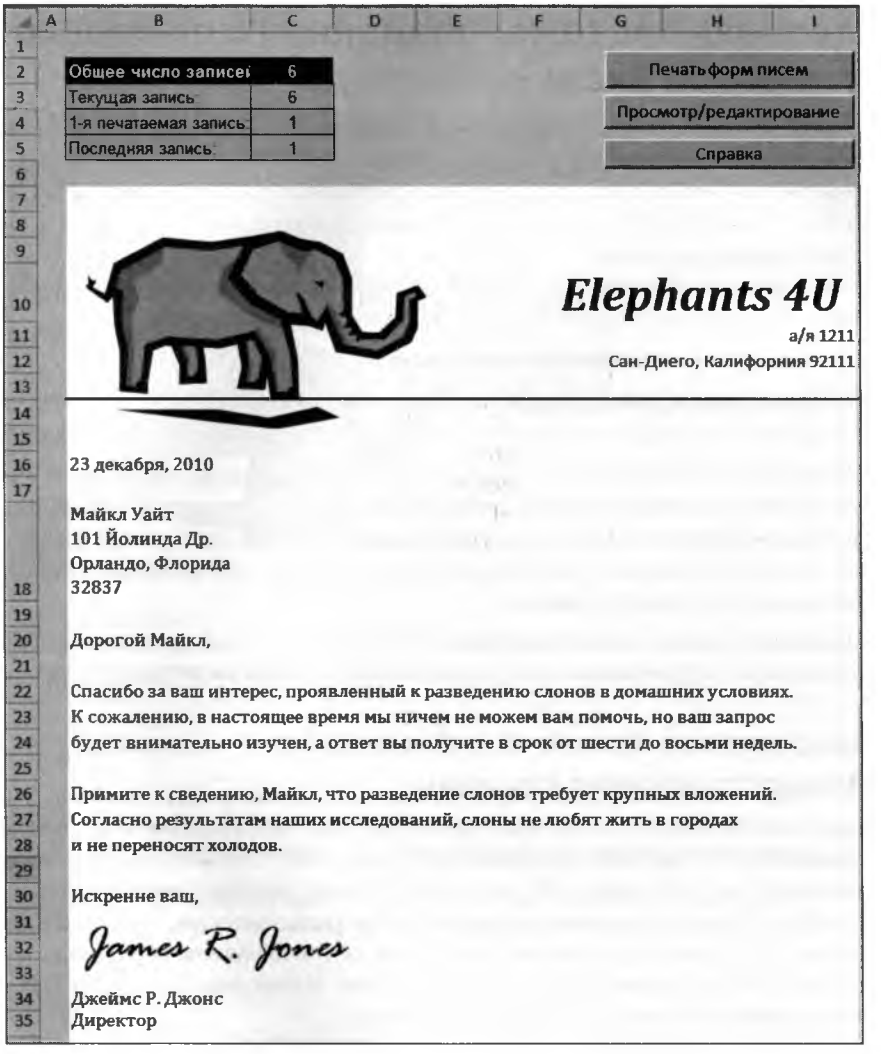
Все примеры, рассматриваемые в этой главе, находятся на прилагаемом к книге компакт-диске. Поскольку большинство примеров включает несколько файлов, каждый пример находится в отдельной папке компакт-диска.

О примерах, приводимых в этой главе

Во многих примерах этой главы используется простая рабочая книга, с помощью которой демонстрируются различные способы получения справочных сведений. Приложение, в качестве которого выступает эта книга, использует хранящиеся на рабочем листе данные для создания и печати писем.

Как можно заметить на следующем рисунке, в ячейках отображается общее количество записей в базе данных (ячейка C2; вычисляется по формуле), номер текущей записи (ячейка C3), первая печатаемая запись (ячейка C4), а также последняя печатаемая запись (ячейка C5). Для отображения конкретной записи следует ввести значение в ячейку C3. Для печати набора писем укажите номера первой и последней записей путем их ввода в ячейки C4 и C5 соответственно.

Приложение очень простое, хотя и включает несколько отдельных компонентов. Это приложение используется для демонстрации различных способов отображения справочной информации.



Рабочая книга включает следующие компоненты.

- `Form (Форма)`. Рабочий лист, который содержит данные приложения.
- `Data (Данные)`. Рабочий лист, который содержит базу данных из семи полей.
- `Help (Справка)`. Этот рабочий лист представлен только в тех примерах, где текст справочной системы сохраняется в рабочих книгах.
- `PrintMod`. Модуль VBA, содержащий макросы печати писем.
- `HelpMod`. Модуль VBA, содержащий макросы, которые отображают текст справочной системы. Содержимое этого модуля изменяется в зависимости от типа отображаемой справки.
- `UserForm1`. Пользовательское диалоговое окно, которое отображается только в том случае, когда способ демонстрации справки предусматривает применение форм `UserForm`.

Справочная система, созданная с помощью компонентов Excel

Возможно, самым простым методом предоставления справочного руководства является использование средств, которые поддерживаются в Excel. Основное преимущество данного метода — необязательность изучения методов создания файлов HTML Help. Ознакомление с последними методами может занять довольно много времени и увеличить срок разработки приложения.

В данном разделе приведен обзор методик предоставления справочной информации, в которых используются следующие встроенные в Excel компоненты.

- **Примечания к ячейкам.** Проще не бывает!
- **Элемент управления `TextBox` (Текстовое поле).** Простой макрос, который отображает на экране текстовое поле, содержащее справочную информацию.
- **Рабочий лист.** Простым способом добавления справочного руководства в приложение является вставка рабочего листа, на котором отображается необходимая справочная информация. После ввода информации рабочий лист можно назвать Справка. Если пользователь щелкнет на ярлычке этого листа, будет активизирована справочная система приложения.
- **Пользовательское диалоговое окно.** Некоторые методы построения справочного руководства подразумевают использование диалоговых окон `UserForm`.

Использование примечаний к ячейке для предоставления справки

Самый простой способ предоставления справочной информации конечному пользователю заключается в добавлении примечаний к ячейкам. Эта методика является наиболее подходящей для описания типа вводимых данных, которые принимаются в определенной ячейке. Когда пользователь наводит указатель мыши на ячейку, содержащую примечание, текст этого примечания отображается в небольшом окне, напоминающем инструментальную подсказку (рис. 24.1). Еще одно преимущество использования этой методики заключается в том, что не нужны макросы.

Помимо этого, возможно автоматическое отображение примечаний к ячейкам. Следующий оператор VBA обеспечивает отображение индикаторов примечаний в ячейках:

```
Application.DisplayCommentIndicator = xlCommentIndicatorOnly
```

Компакт-диск

Рабочая книга, демонстрирующая использование примечаний, находится на прилагаемом компакт-диске в файле `cell comments\formletter.xlsm`.

Совет

Большинство пользователей даже не подозревает о том, что примечания могут отображать изображения. Щелкните правой кнопкой мыши на границе примечания, затем в контекстном меню выберите команду Формат примечания (Format Comment). В диалоговом окне Формат примечания (Format Comment) выберите вкладку Цвета и линии (Colors and Lines). Щелкните на раскрывающемся списке Цвет (Color), после чего выберите пункт Способы

заливки (Fill Effects). В диалоговом окне Способы заливки (Fill Effects) выберите вкладку Рисунок (Picture) и щелкните на кнопке Рисунок (Select Picture) для выбора файла изображения.

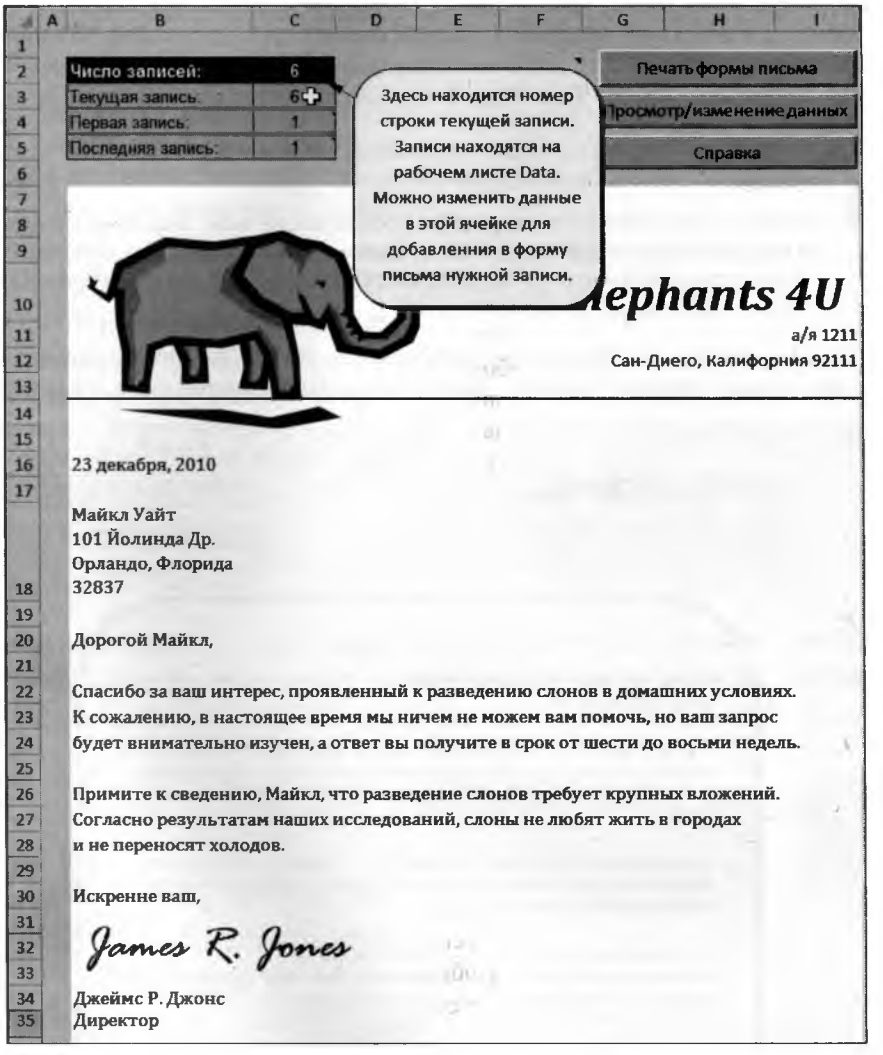


Рис. 24.1. Использование примечаний к ячейкам в качестве справки

Вы также можете воспользоваться командой Excel Данные⇨Работа с данными⇨Проверка данных (Data⇨Data Tools⇨Data Validation), которая отображает диалоговое окно, где можно указать критерии проверки данных, вводимых в ячейку или диапазон. Можно игнорировать возможности по проверке данных, обратившись к вкладке Сообщение для ввода (Input Message) диалогового окна Проверка вводимых значений (Data Validation) для указания сообщения, которое отображается при активизации ячейки. Максимальная длина этого сообщения может составлять 250 символов.

Применение текстового поля для предоставления справки

Использование элемента управления `TextBox` для отображения справочной информации является еще одним простым в реализации методом предоставления справочной информации пользователю. Создайте текстовое поле путем выполнения команды Вставка⇨Текст⇨Надпись (Insert⇨Text⇨Text Box), введите текст справки и отформатируйте его требуемым образом.

Совет

Вместо текстового поля можно воспользоваться различными фигурами, в которые следует добавить текст справки. Сначала выберите команду Вставка⇨Иллюстрации⇨Фигуры (Insert⇨Illustrations⇨Shapes), а затем нужную фигуру. После этого введите нужный текст.

На рис. 24.2 показан пример фигуры, применяемой для отображения справочной информации. Для создания эффекта "плавания" объекта над рабочим листом была добавлена тень.

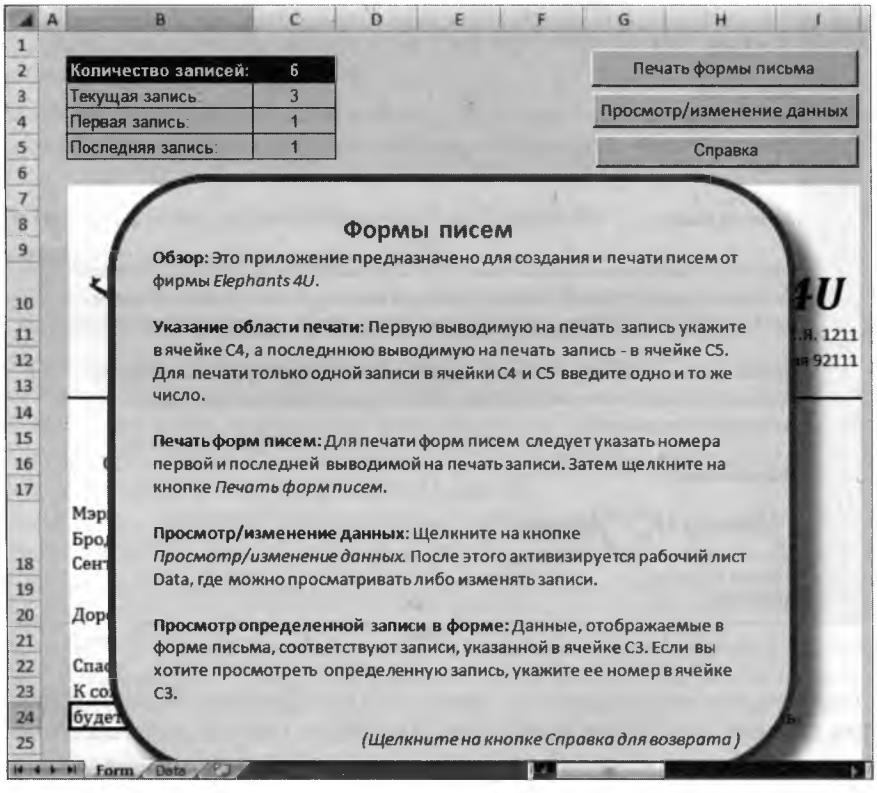


Рис. 24.2. Использование фигуры для отображения справочной информации

Как правило, отображать текстовое поле, используемое в качестве справки, нужно лишь эпизодически. Поэтому целесообразно добавить в приложение кнопку, которая вызывает макрос, присваивающий значение свойству `Visible` текстового поля. Ниже

представлен пример подобного макроса. В данном случае текстовому полю TextBox присвоено имя HelpText.

```
Sub ToggleHelp()
   ActiveSheet.TextBoxes("HelpText").Visible = _
      Not ActiveSheet.TextBoxes("HelpText").Visible
End Sub
```

Компакт-диск

Рабочая книга, демонстрирующая использование текстового поля для отображения справки, находится на прилагаемом компакт-диске в файле textbox\formletter.xlsm.

Использование рабочего листа для отображения справки

Еще один способ добавления справочной системы в создаваемые приложения — создание макроса, который активизирует отдельный рабочий лист, включающий справочные сведения. Просто свяжите макрос с кнопкой, кнопкой панели инструментов либо элементом меню и... "дело в шляпе".

На рис. 24.3 показан пример рабочего листа, отображающего справку. Был разработан диапазон, который содержит текст справки. Причем имитируется окрашенный в желтый цвет лист блокнота, который традиционно используется для создания заметок.

Для предотвращения прокрутки листа справки макрос присваивает соответствующее значение свойству ScrollArea рабочего листа. Поскольку значение свойства не сохраняется вместе с рабочей книгой, следует устанавливать его после активизации рабочего листа. Рабочий лист был защищен таким образом, что предотвращается изменение текста и выделение ячеек пользователем. Также была "заморожена" первая строка, в результате чего кнопка Return (Возврат) всегда видима, независимо от того, насколько далеко была выполнена прокрутка листа.

Основной недостаток, связанный с применением описанного выше метода предотвращения прокрутки листа справки, заключается в том, что текст справки будет невидим (наравне с основной рабочей областью). Одно из возможных решений этой проблемы заключается в создании макроса, открывающего новое окно, в котором отображается лист.

Компакт-диск

На прилагаемом компакт-диске находится рабочая книга (worksheet\formletter.xlsm), демонстрирующая использование рабочего листа для отображения справки.

Отображение справки в пользовательском диалоговом окне

Еще один способ предоставления справки пользователям — отображение ее текста в пользовательском диалоговом окне. В этом разделе будут описаны соответствующие

Рис. 24.3. Размещение справки на отдельном рабочем листе не составит особого труда

Использование элементов управления `Label` для отображения текста справочного руководства

На рис. 24.4 показано диалоговое окно `UserForm`, которое содержит два элемента управления `Label`: один элемент управления используется для отображения заголовка, а второй — для отображения самого текста. Элемент управления `SpinButton` позволяет пользователю перемещаться по разделам справочного руководства. Сам текст сохраняется на рабочем листе. Названия разделов справочного руководства хранятся в столбце A, а текст разделов — в столбце B.

После щелчка на элементе управления `SpinButton` вызывается указанная ниже процедура. Эта процедура устанавливает значение `Caption` для двух элементов управления `Label` таким образом, что оно будет соответствовать тексту в определенной строке рабочего листа под названием `HelpSheet` (Справка).

```
Private Sub SpinButton1_Change()
   HelpTopic = SpinButton1.Value
   LabelTopic.Caption = Sheets("HelpSheet"). _
      Cells(HelpTopic, 1)
   LabelText.Caption = Sheets("HelpSheet").Cells(HelpTopic, 2)
   Me.Caption = APPNAME & " (Help Topic " & HelpTopic & " of " _
```

```
      & SpinButton1.Max & ")"
End Sub
```

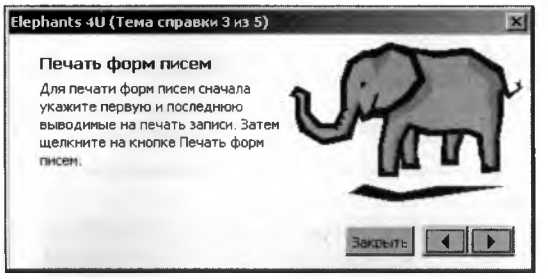


Рис. 24.4. Щелчок на одной из стрелок элемента управления `SpinButton` изменяет текст подписи

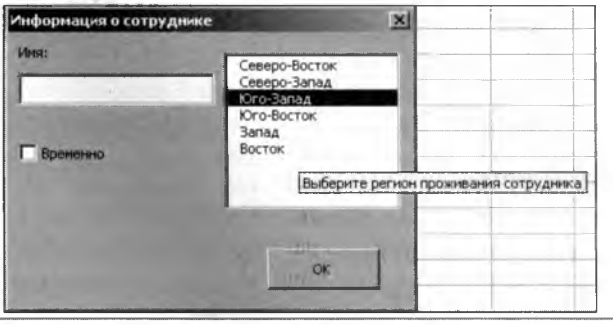
В данном случае `APPNAME` — это глобальная константа, которая содержит название приложения.

Компакт-диск

Рассматриваемый в этом разделе пример находится на прилагаемом к книге компакт-диске в файле `userform1\formletter.xlsm`.

Использование подсказок элементов управления в пользовательском диалоговом окне

Каждый элемент управления `UserForm` имеет свойство `ControlTipText`, которое может содержать краткий описательный текст. Как только над этим элементом управления устанавливается указатель мыши, во всплывающем окне отображается подсказка (при ее наличии). Ниже представлен соответствующий рисунок.



Использование "прокручиваемого" элемента `Label` для отображения текста справочного руководства

Данная методика заключается в отображении текста справочного руководства с помощью единственного элемента управления `Label`. По причине того, что элемент управления `Label` не может содержать вертикальной полосы прокрутки, он размещается внутри элемента управления `Frame`, поддерживающего использование обеих полос прокрутки. На рис. 24.5 показан пример применения диалогового окна `UserForm`, настроенного описанным выше способом. Пользователь может прокручивать текст с помощью полосы прокрутки элемента управления `Frame`.

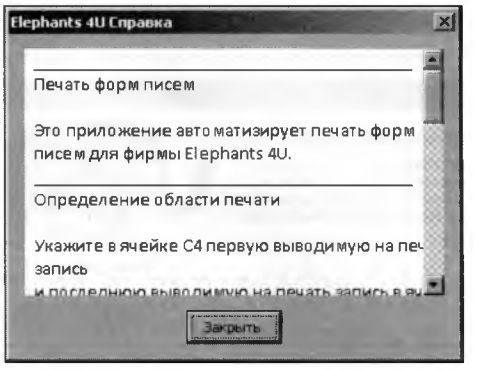


Рис. 24.5. Вставка элемента управления `Label` в элемент управления `Frame` приводит к добавлению полос прокрутки

Текст, который отображается в элементе управления `Label`, получен из рабочего листа `HelpSheet` (Справка). Передача текста осуществляется на этапе инициализации диалогового окна `UserForm`. Ниже приводится код процедуры `UserForm_Initialize` для данного рабочего листа.

```
Private Sub UserForm_Initialize()
   Dim LastRow As Long
   Dim r As Long
   Dim txt As String
   Me.Caption = APPNAME & " Help"
   LastRow = Sheets("HelpSheet").Cells _
      (Rows.Count, 1).End(xlUp).Row
   txt = ""
   For r = 1 To LastRow
      txt = txt & Sheets("HelpSheet").Cells(r, 1) _
         .Text & vbCrLf
   Next r
   With Label1
      .Top = 0
      .Caption = txt
      .Width = 260
      .AutoSize = True
   End With
   With Frame1
      .ScrollHeight = Label1.Height
      .ScrollTop = 0
   End With
End Sub
```

Обратите внимание на то, что код настраивает свойство `ScrollHeight` объекта `Frame` таким образом, чтобы в процессе прокрутки охватывалась полная высота элемента управления `Label`. И снова, `APPNAME` — это глобальная константа, которая содержит имя приложения.

Поскольку элемент управления `Label` не может отображать форматированный текст, для выделения заголовков разделов справки применяется подчеркивание символов в рабочем листе `HelpSheet` (Справка).

Компакт-диск

Рабочая книга, демонстрирующая описанную методику, находится на прилагаемом компакт-диске в файле `userform2\formletter.xlsm`.

Использование раскрывающегося списка для выбора раздела справочного руководства

Пример, приведенный в этом разделе, расширит возможности предыдущего примера. На рис. 24.6 показано диалоговое окно `UserForm`, которое содержит элементы управления `ComboBox` и `Label`. Пользователь может выбрать раздел справочного руководства из раскрывающегося списка, который предоставляется элементом управления `ComboBox`, а также последовательно просмотреть разделы с помощью кнопок Назад и Вперед.

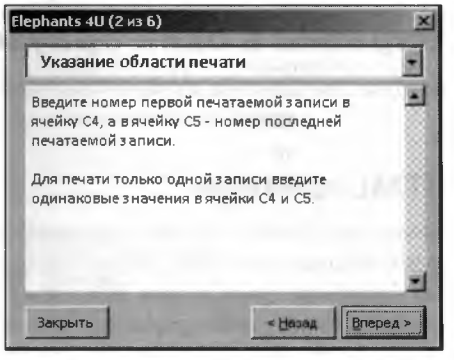


Рис. 24.6. Использование раскрывающегося списка для выбора раздела справки

Приведенный пример немного сложнее, чем пример из предыдущего раздела, но он предоставляет большую гибкость. В данном случае используется методика "Label Frame" (она рассматривалась ранее), что позволяет вводить разделы справочного руководства любой длины.

Текст справочного руководства хранится в рабочем листе, который называется `HelpSheet` (Справка). Он расположен в двух столбцах: A и B. Первый столбец содержит названия разделов справочного руководства, а второй — текст разделов. Опции элемента управления `ComboBox` добавляются во время выполнения процедуры `UserForm_Initialize`. Переменная `CurrentTopic` определена на уровне модуля и содержит целое число, которое идентифицирует выбранный раздел справочного руководства.

```
Private Sub UpdateForm()
   ComboBoxTopics.ListIndex = CurrentTopic - 1
   Me.Caption = HelpFormCaption & _
      " (" & CurrentTopic & " of " & TopicCount & ")"

   With LabelText
      .Caption = HelpSheet.Cells(CurrentTopic, 2)
      .AutoSize = False
      .Width = 212
      .AutoSize = True
   End With
   With Frame1
      .ScrollHeight = LabelText.Height + 5
      .ScrollTop = 1
   End With
```

```
    If CurrentTopic = 1 Then
       NextButton.SetFocus
    ElseIf CurrentTopic = TopicCount Then
       PreviousButton.SetFocus
    End If
    PreviousButton.Enabled = CurrentTopic <> 1
    NextButton.Enabled = CurrentTopic <> TopicCount
End Sub
```

Компакт-диск

Рабочая книга, демонстрирующая описанную методику, находится на прилагаемом компакт-диске в файле `userform3\formletter.xlsm`.

Отображение справки в окне браузера

В этом разделе описываются два способа отображения пользовательской справки в окне браузера.

Использование HTML-файлов

Еще один способ отображения справки, предназначенной для приложения Excel, — создание одного или большего количества HTML-файлов, а также формирование гиперссылки, которая отображает подобный файл в окне браузера, заданного по умолчанию. Файлы HTML могут храниться на локальном диске или в корпоративной сети. Можно также в ячейке создать ссылку на файл справки (при этом не нужен макрос). На рис. 24.7 показан пример отображения справки в окне браузера.

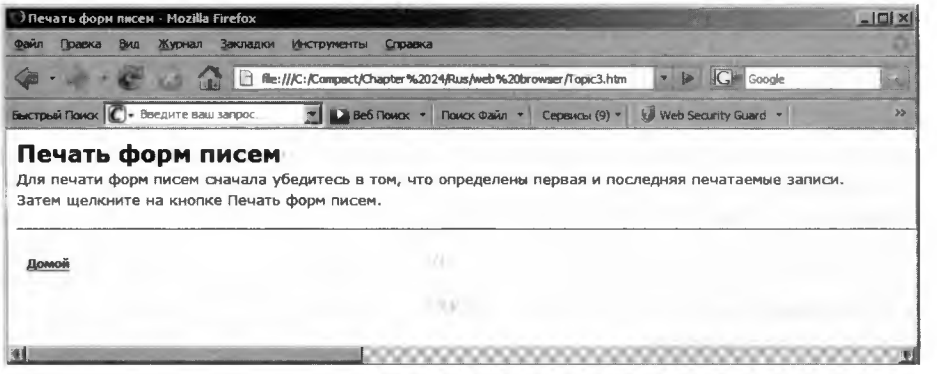


Рис. 24.7. Отображение справки в окне браузера

Компакт-диск

Рабочая книга, демонстрирующая описанную методику, находится на прилагаемом компакт-диске в файле `web browser\formletter.xlsm`.

Использование файла MHTML

Аббревиатура MHTML образована от слов MIME Hypertext Markup Language и обозначает формат сжатия, используемый в Интернете. Файлы MHTML могут отображаться с помощью Microsoft Internet Explorer.

Хорошая новость состоит в том, что файлы MHTML можно создавать в Excel. Для этого начните с создания текста справки, разместив его на произвольном количестве рабочих листов. После этого выберите команду **Файл⇨Сохранить как** (File⇨Save As), щелкните на раскрывающемся списке **Тип файла** (Save As Type) и выберите пункт **Веб-страница в одном файле** (Single File Web Page (*.mht; *.mhtml)). Учтите, что в этом формате не могут сохраняться макросы VBA.

На рис. 24.8 показан MHTML-файл, отображенный в окне Internet Explorer.

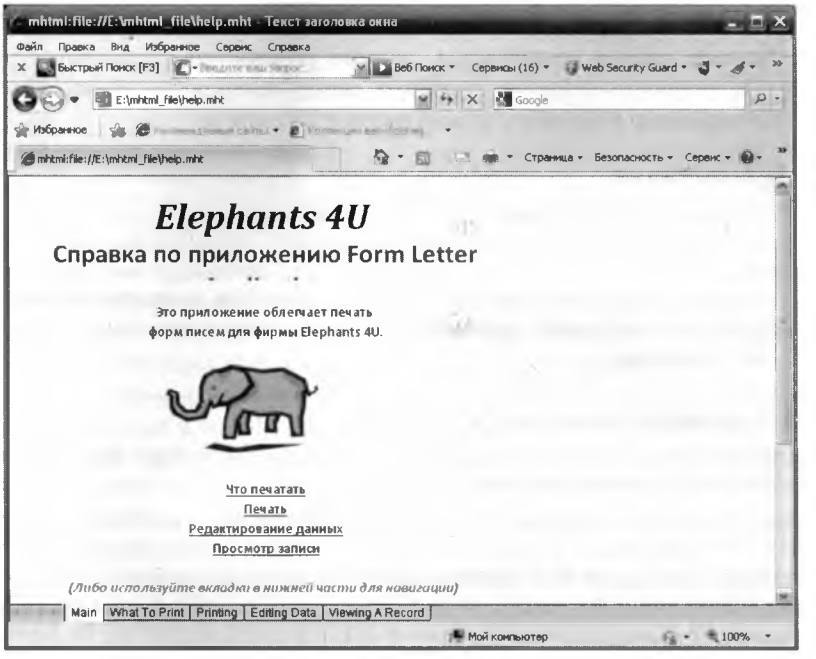


Рис. 24.8. Отображение MHTML-файла в окне веб-браузера

Предупреждение

Некоторые версии Internet Explorer не могут отображать MHTML-файлы, созданные с помощью Excel, если имя файла либо путь включают пробелы.

В Excel можно создать гиперссылку в целях отображения MHTML-файла.

Компакт-диск

Рабочая книга, демонстрирующая описанную технику, находится на прилагаемом компакт-диске в файле `mhtml file\formletter.xlsm`.

Примечание

Если состоящая из нескольких листов рабочая книга Excel сохранена в виде MHTML-файла, этот файл может содержать код Javascript. При открытии подобного файла может отображаться предупреждение подсистемы безопасности.

Использование средства HTML Help

В настоящее время в приложениях Windows чаще всего используются справочные руководства на основе технологии HTML Help, с помощью которой можно управлять файлами справочной системы в формате CHM. Эта технология заменила старую технологию WinHelp с файлами формата HLP (см. врезку "Эволюция справочных систем Microsoft"). Обе справочные системы позволяют разработчикам связывать идентификатор контекста с определенным разделом справочной системы. Благодаря этому можно отображать определенный раздел справки контекстно-зависимым образом.

В этом разделе кратко описывается система разработки справочных руководств HTML Help. Подробности создания подобных систем выходят за рамки данной книги. Требуемую информацию и примеры можно найти в Интернете.

Примечание

Если вы планируете разрабатывать полномасштабную справочную систему, я настоятельно рекомендую вам приобрести специализированное ПО, которое значительно облегчит разработку. Это связано с тем, что специализированные программы берут на себя выполнение тяжелой и нудной работы. Вы можете выбирать среди бесплатных, условно бесплатных и коммерческих программ.

Эволюция справочных систем Microsoft

На протяжении ряда лет компания Microsoft внедрила четыре версии справочных систем в своих операционных системах и приложениях.

- **WinHelp.** Основана на файлах RTF (Rich Text Formatting — расширенный текстовый формат). Эта справочная система впервые была использована в Windows 3.0 (в 1990 году). Несколько RTF-файлов можно скомпилировать в один файл справки с расширением `.hlp`. Эта справочная система использовалась в версиях Microsoft Office, предшествующих Office 2000.
- **HTML Help.** Основана на HTML-файлах (HyperText Markup Language — язык гипертекстовой разметки). Эта система впервые была использована в Internet Explorer 4.0 (в 1997 году). Несколько HTML-файлов можно скомпилировать в один файл справки с расширением `.chm`. Первой версией Office, в которой использовалась эта справочная система, была Office 2000.
- **Microsoft Help 2.** Поддерживает HTML, DHTML, XML, VBScript и JavaScript. Несколько файлов можно скомпилировать в один файл справки с расширением `.hsx`. Эта справочная система используется в Office 2007, а также в ряде других полномасштабных приложений.
- **Assistance Platform Help.** Эта справочная система используется в Windows Vista и Windows 7.

Скомпилированная система HTML Help преобразует набор отдельных HTML-файлов в компактную справочную систему. Помимо этого можно создавать комбинированное содержание и предметный указатель, а также гипертекстовую систему поиска на основе ключевых слов. Система HTML Help может также использовать дополнительные средства, например графические файлы, элементы управления ActiveX, сценарии и код DHTML (Dynamic HTML — Динамический HTML). На рис. 24.9 показан простой пример системы HTML Help.

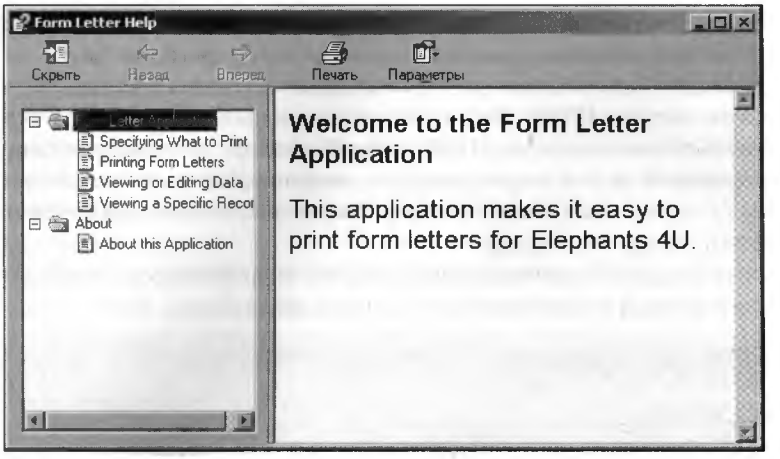


Рис. 24.9. Пример системы HTML Help

Отображение раздела справки Excel Help

Иногда возникает необходимость в отображении определенного раздела системы Excel Help с помощью кода VBA. Например, требуется получить сведения о типах диаграмм в Excel.

Вначале убедитесь в том, что справочная система Excel отображает справку с локального компьютера, а не с веб-сайта Office.com. (Воспользуйтесь элементом управления, находящимся в правом нижнем углу окна справки, для изменения состояния подключения к Интернету.) На первом этапе нужно определить идентификатор `Topic ID` для данного раздела. Для этого найдите раздел в справочной системе, щелкните правой кнопкой мыши и в контекстном меню выберите команду Свойства (Properties). В поле Адрес (URL-ссылка) (Address (URL)) находится идентификатор `Topic ID`, внедренный в URL-ссылку (состоящая из 10 символов строка, которая начинается литерой `H`). Скопируйте идентификатор `Topic ID`, а затем используйте его в следующем операторе VBA:

```
Application.Assistance.ShowHelp " HA10342187"
```

В Excel 2007 это работает только в том случае, если пользовательская справочная система отображает лишь локальный контекст (т.е. справочная система находится в автономном режиме). В Excel 2010 эти ограничения не действуют.

Можно также воспользоваться методом `SearchHelp`. Для этого метода достаточно указать выражение поиска, после чего на экране отобразится перечень связанных разделов справки. Ниже приведен соответствующий пример.

```
Application.Assistance.SearchHelp "формат элементов диаграммы "
```

Компакт-диск

Рабочая книга, демонстрирующая описанную технологию, находится на прилагаемом компакт-диске в файле `html help\formletter.xlsm`.

Система HTML Help использует для отображения данных программу HTML Help Viewer, которая создана на основе ядра браузера Internet Explorer. Информация отображается в основном окне, а содержание, указатель и инструменты поиска — на отдельной панели. Текст справки может также содержать стандартные гиперссылки, с помощью которых осуществляется отображение другого раздела справки или документа, находяще-

гося в Интернете. Система HTML Help может также получать доступ к файлам, находящимся на веб-сайте. Благодаря этому пользователь может получать самую свежую справочную информацию, которой еще не было на момент создания системы HTML Help.

Для создания системы HTML Help нужен специальный компилятор. Например, можно воспользоваться компилятором HTML Help Workshop, который, наравне с дополнительной информацией по его использованию, свободно доступен на веб-сайте Microsoft MSDN (`http://msdn.microsoft.com`). Откройте этот сайт и в строке поиска введите фразу `HTML Help Workshop`.

На рис. 24.10 показано окно компилятора HTML Help Workshop с файлом проекта, на основе которого создана справочная система, показанная на рис. 24.9.

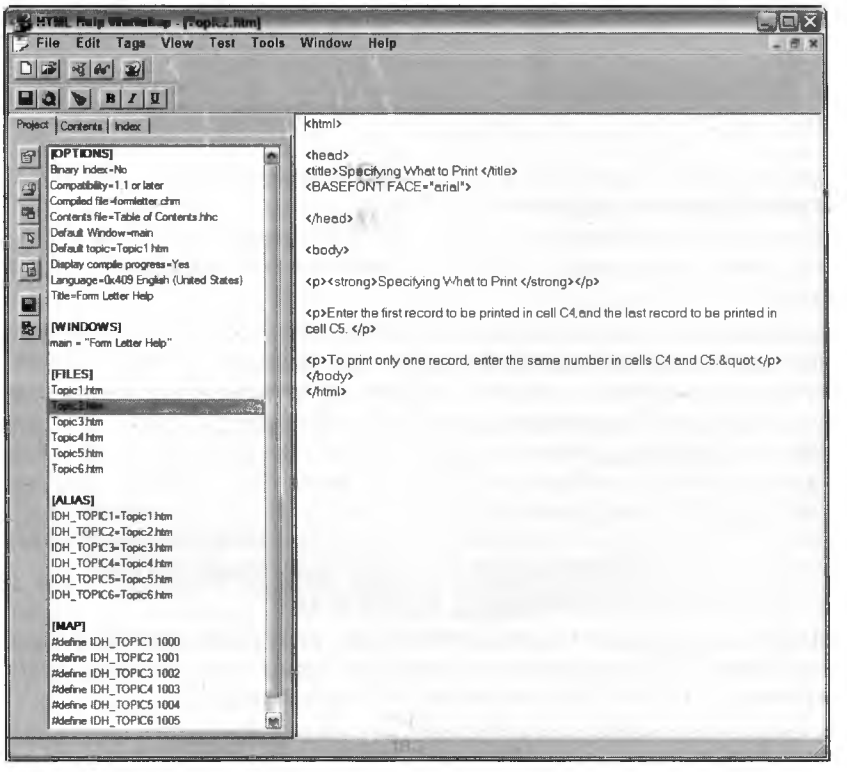


Рис. 24.10. С помощью приложения HTML Help Workshop можно создать файл справки

Метод `Help`

Воспользуйтесь методом `Help` объекта `Application` для отображения файла справочного руководства. Этот файл может иметь как формат WinHelp HLP, так и формат HTML Help CHM. Данный метод работает даже тогда, когда для файла не определены идентификаторы раздела.

Метод `Help` имеет следующий синтаксис:

```
Application.Help(helpFile, helpContextID)
```

Оба аргумента являются необязательными. Если имя файла справочного руководства не указано, то будет отображаться файл справочного руководства Excel. Если опустить идентификатор раздела, то указанный файл будет отображен с использованием раздела, принятого по умолчанию.

Следующий пример отображает раздел файла `myapp.chm`, принятый по умолчанию. Файл должен находиться в той же папке, что и рабочая книга приложения, из которого он вызывается. Обратите внимание: второй аргумент не указан.

```
Sub ShowHelpContents()
   Application.Help ThisWorkbook.Path & "\myapp.chm"
End Sub
```

Представленный далее оператор отображает раздел справочного руководства с идентификатором 1002. При этом используется файл справочного руководства в формате HTML Help, который называется `myapp.chm`.

```
Application.Help ThisWorkbook.Path & "\myapp.chm", 1002
```

Связывание файлов справочного руководства с приложением

Файл справочного руководства можно связать с приложением одним из двух способов: с помощью диалогового окна Project Properties (Свойства проекта) или посредством создания специального кода VBA.

Откройте окно редактора Visual Basic Editor и выберите команду Tools⇨*xxx* Properties (Сервис⇨Свойства *xxx*), где *xxx* соответствует имени проекта. В диалоговом окне Project Properties (Свойства проекта) перейдите на вкладку General (Общие) и укажите скомпилированный файл справки HTML Help для проекта. Этот файл должен иметь расширение `.chm`.

Представленный далее оператор связывает приложение с файлом `myfuncs.chm`. Этот файл находится в той же папке, что и файл рабочей книги.

```
ThisWorkbook.VBProject.HelpFile = ThisWorkbook.Path & "\myfuncs.chm"
```

Примечание

Если в результате выполнения указанного выше оператора появляется сообщение об ошибке, следует открыть программный доступ к проектам VBA. В Excel выполните команду Разработчик⇨Код⇨Безопасность макросов (Developer⇨Code⇨Macro Security) для отображения диалогового окна Центр управления безопасностью (Trust Center). Затем отмените установку флажка Доверять доступ к объектной модели проектов VBA (Trust Access to the VBA Project Object Model).

После связывания файла справочного руководства с приложением можно отобразить определенные справочные сведения в следующих ситуациях.

- Когда пользователь нажимает клавишу <F1> при выборе пользовательской функции в диалоговом окне Мастер функций (Insert function).
- Когда пользователь нажимает клавишу <F1> при отображенном диалоговом окне `UserForm`. В результате отображается раздел справочного руководства, который связан с элементом управления, активным в момент нажатия клавиши <F1>.

Связывание раздела справочного руководства с функцией VBA

Если с помощью кода VBA создаются новые функции рабочего листа, то может возникнуть необходимость связать файл справочного руководства и идентификатор раздела с каждой функцией. Как только связь между этими элементами будет установлена, раздел справочного руководства сможет быть отображен после нажатия клавиши <F1> в диалоговом окне Мастер функций.

Для того чтобы указать идентификатор раздела для новой функции рабочего листа, следуйте приведенным ниже инструкциям.

1. Создайте функцию.
2. Удостоверьтесь, что с проектом связан файл справочного руководства (для получения дополнительной информации обратитесь к предыдущему разделу).
3. В окне редактора Visual Basic Editor нажмите клавишу <F2> для активизации окна Object Browser.
4. Укажите проект в раскрывающемся списке Project/Library (Проект/библиотека).
5. В списке Classes (Классы) выберите модуль, который содержит функцию.
6. В окне Member (Компонент) укажите функцию.
7. Щелкните на функции правой кнопкой мыши и выберите Properties (Свойства) из появившегося контекстного меню. Таким образом, будет отображено диалоговое окно Member Options (Свойства компонента), как показано на рис. 24.11.

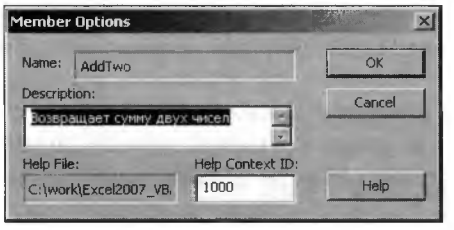


Рис. 24.11. Определите идентификатор для пользовательской функции

Введите идентификатор раздела справочного руководства, который связан с этой функцией. Вы также можете добавить описание функции.

Примечание

Диалоговое окно Member Options не позволяет указать новый файл справочного руководства. Это диалоговое окно всегда использует новый файл, который был ранее связан с проектом.

Может оказаться, что проще создать код VBA, который будет настраивать идентификатор раздела и файл справочного руководства для новых функций. Для этого воспользуйтесь методом `MacroOptions`.

Следующая процедура использует метод `MacroOptions` для определения описания функции, применяемого файла справочного руководства и идентификатора раздела для двух функций (`AddTwo` и `Squared`). Этот макрос достаточно вызвать на выполнение один раз.

```
Sub SetOptions()
'  Установить параметры для функции AddTwo
   Application.MacroOptions Macro:="AddTwo", _
      Description:="Сумма двух чисел", _
      HelpFile:=ThisWorkbook.Path & "\myfuncs.chm", _
      HelpContextID:=1000
      ArgumentDescriptions:=Arrays("Первое число в сумме", _
         "Второе число в сумме")

'  Установка параметров для функции Squared
   Application.MacroOptions Macro:="Squared", _
      Description:="Квадрат от значения аргумента ", _
      HelpFile:=ThisWorkbook.Path & "\myfuncs.chm", _
      HelpContextID:=2000
      ArgumentDecriptions:=Array("Число возведено в квадрат")
End Sub
```

После запуска этих процедур на выполнение пользователь может получить справку непосредственно в диалоговом окне Мастер функций (Insert Function), нажав клавишу <F1> либо щелкнув на ссылке Справка по этой функции (Help on This Function).

Новинка

В предыдущем примере было продемонстрировано использование нового аргумента метода `MacroOptions`. В частности, в Excel 2010 поддерживается аргумент `ArgumentDescriptions`. С помощью этого аргумента можно описывать все аргументы функции. Эти описания отображаются в диалоговом окне Аргументы функции (Function Arguments), которое вызывается из диалогового окна Мастер функций (Insert Function).

Компакт-диск

Демонстрирующая описанную технологию рабочая книга находится на прилагаемом компакт-диске в файле `function help\myfuncs.xlsm`.

Глава 25

Разработка пользовательских приложений

В этой главе...

- Что такое приложение, ориентированное на пользователя
- Мастер расчета займа
- Концепции разработки приложений

В данной главе описывается приложение, ориентированное на конечного пользователя, — мастер расчета займа. Это приложение нашло практическое применение, хотя мы будем использовать его для изучения методов разработки приложений, ориентированных на пользователя.

Что такое приложение, ориентированное на пользователя

Приложение, ориентированное на пользователя, — это приложение Excel, которое можно использовать без специальной подготовки. Такие приложения позволяют получить конечный результат даже тем пользователям, которые совершенно ничего не знают об Excel.

Приложение мастера расчета займа, которое рассматривается в этой главе, относится к приложениям, ориентированным на пользователя (оно разрабатывалось специально для того, чтобы помочь пользователям, которые совершенно ничего не знают о функционировании Excel). Ответив на несколько простых вопросов, вы сможете создать полезный и гибкий рабочий лист, содержащий необходимые формулы.

Мастер расчета займа

Приложение мастера расчета займа (Loan Amortization Wizard) создает рабочий лист, который содержит график погашения кредита с фиксированной процентной ставкой. Этот график представлен в виде информации о ежемесячных выплатах. Таким образом, отображаются сведения о сумме ежемесячных выплат, а также данные о том, какая часть суммы идет на погашение процентов, а какая — на погашение основной суммы займа. Более того, пользователь может ознакомиться и с информацией о новом балансе.

Альтернативой такого приложения можно считать файл шаблона (`*.XLT`) с необходимыми формулами. Далее вы убедитесь, что приложение в форме мастера предоставляет несколько дополнительных преимуществ.

На рис. 25.1 показан график погашения кредита, который создан приложением мастера расчета займа.

График погашения кредита

Подготовлено Александром Сергеевым

Сгенерировано 23 декабря 2010 г.

Возвращаемая сумма:	378 000 рублей
Первый взнос в процентах:	20,0%
Первый взнос:	75 600 рублей
Сумма кредита:	302 400 рублей
Срок кредита (месяцы):	360
Процентная ставка:	6,25%
Срок первого платежа:	01.03.2010

Номер платежа	Год	Месяц	Платеж	Проценты	Тело кредита	Баланс
1	2010	3	1 862 рублей	1 575 рублей	287 рублей	302 113 рублей
2	2010	4	1 862 рублей	1 574 рублей	288 рублей	301 825 рублей
3	2010	5	1 862 рублей	1 572 рублей	290 рублей	301 535 рублей
4	2010	6	1 862 рублей	1 570 рублей	291 рублей	301 243 рублей
5	2010	7	1 862 рублей	1 569 рублей	293 рублей	300 950 рублей
6	2010	8	1 862 рублей	1 567 рублей	294 рублей	300 656 рублей
7	2010	9	1 862 рублей	1 566 рублей	296 рублей	300 360 рублей
8	2010	10	1 862 рублей	1 564 рублей	298 рублей	300 062 рублей
9	2010	11	1 862 рублей	1 563 рублей	299 рублей	299 763 рублей
10	2010	12	1 862 рублей	1 561 рублей	301 рублей	299 463 рублей
2010 Итог			***18 619 рублей***	***15 682 рублей***	***2 937 рублей***	***299 463 рублей***
11	2011	1	1 862 рублей	1 560 рублей	302 рублей	299 160 рублей
12	2011	2	1 862 рублей	1 558 рублей	304 рублей	298 856 рублей
13	2011	3	1 862 рублей	1 557 рублей	305 рублей	298 551 рублей
14	2011	4	1 862 рублей	1 555 рублей	307 рублей	298 244 рублей
15	2011	5	1 862 рублей	1 553 рублей	309 рублей	297 936 рублей
16	2011	6	1 862 рублей	1 552 рублей	310 рублей	297 625 рублей
17	2011	7	1 862 рублей	1 550 рублей	312 рублей	297 314 рублей
18	2011	8	1 862 рублей	1 549 рублей	313 рублей	297 000 рублей
19	2011	9	1 862 рублей	1 547 рублей	315 рублей	296 685 рублей
20	2011	10	1 862 рублей	1 545 рублей	317 рублей	296 368 рублей
21	2011	11	1 862 рублей	1 544 рублей	318 рублей	296 050 рублей
22	2011	12	1 862 рублей	1 542 рублей	320 рублей	295 730 рублей
2011 Итог			***22 343 рублей***	***18 611 рублей***	***3 732 рублей***	***295 730 рублей***
23	2012	1	1 862 рублей	1 540 рублей	322 рублей	295 408 рублей

Рис. 25.1. Этот график наглядно знакомит с выплатами ипотечного кредита, выданного на 30 лет

Компакт-диск

Это приложение находится на прилагаемом к книге компакт-диске в файле `loan amortization wizard.xlam`, представляющем собой незащищенную надстройку.

Использование мастера

Приложение мастера расчета займа отображает последовательность из пяти диалоговых окон, которые запрашивают информацию у пользователя. Как и другие мастера, это приложение позволяет перемещаться вперед и назад по этапам работы мастера. Щелчок на кнопке Готово приведет к созданию нового рабочего листа. Если после щелчка на кнопке Готово обнаруживается, что не были пройдены все этапы, используются значения, заданные по умолчанию. После щелчка на кнопке Отмена пользовательское диалоговое окно закрывается, и действия не выполняются.

Данное приложение использует единственное диалоговое окно `UserForm` с элементом управления `MultiPage`, которое содержит вкладки для всех пяти этапов работы мастера (рис. 25.2–25.6).

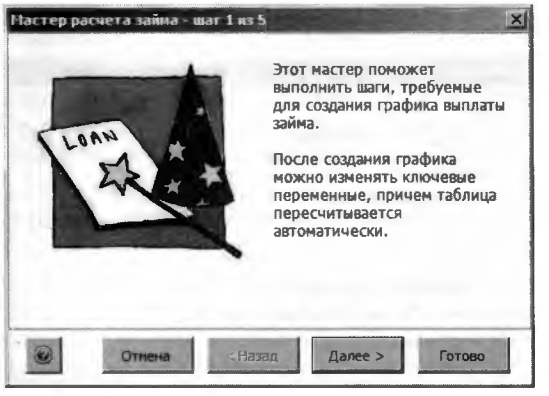


Рис. 25.2. Шаг 1 мастера расчета займа

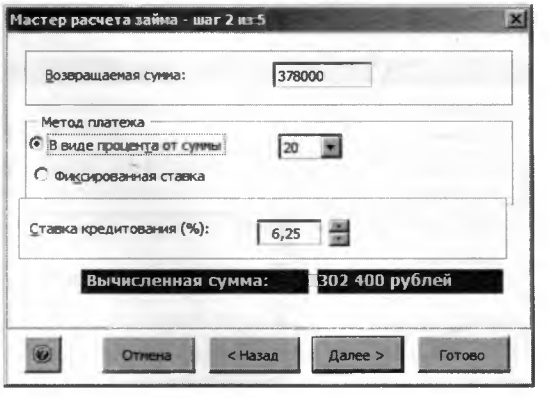


Рис. 25.3. Шаг 2 мастера расчета займа

Структура рабочей книги

Приложение мастера расчета займа состоит из следующих компонентов:

- `FormMain` — диалоговое окно `UserForm`, которое служит основным пользовательским интерфейсом;

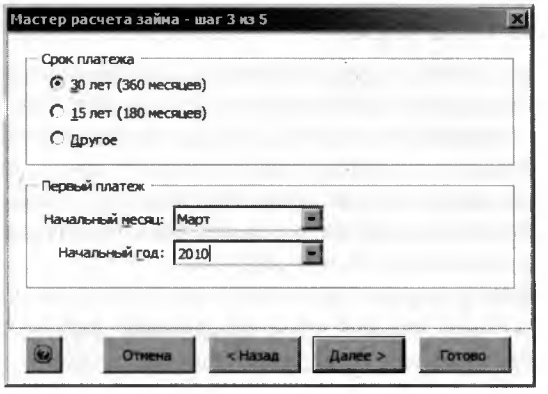


Рис. 25.4. Шаг 3 мастера расчета займа

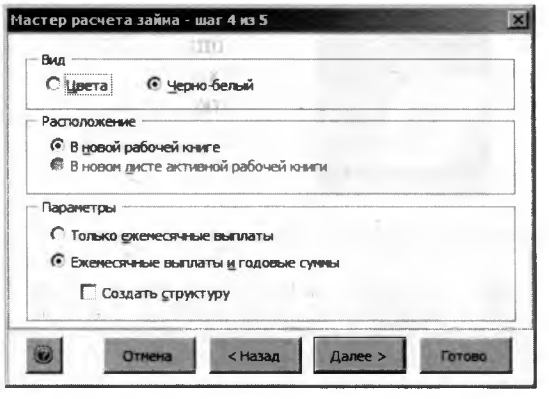


Рис. 25.5. Шаг 4 мастера расчета займа

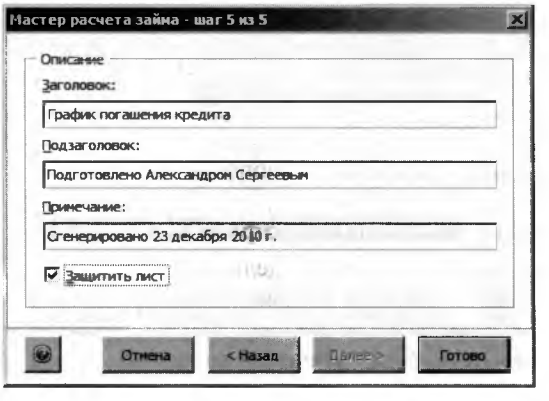


Рис. 25.6. Шаг 5 мастера расчета займа

- `FormHelp` — диалоговое окно `UserForm`, которое используется для отображения интерактивного справочного руководства;
- `HelpSheet` — рабочий лист, который содержит текст, отображаемый в интерактивном справочном руководстве;

- `ModMain` — модуль VBA, который содержит процедуру, отображающую основное диалоговое окно `UserForm`;
- `ThisWorkbook` — в этом модуле кода находятся процедуры обработки событий `Open` и `BeforeClose` объекта `Workbook`, позволяющие создавать и удалять необходимые меню.

Также файл рабочей книги содержит простой XML-код RibbonX, который на ленте создает кнопку Loan Amortization Wizard (Мастер расчета займа).

Как это работает

Мастер расчета займа представляет собой надстройку, для установки которой используется диалоговое окно **Надстройки** (Add-Ins). Для отображения этого диалогового окна выберите команду **Файл⇨Параметры Excel⇨Надстройки** (File⇨Excel Options⇨Add-Ins). В диалоговом окне **Параметры Excel** (Excel Options) в списке **Управление** (Manage) выберите пункт **Надстройки Excel** (Excel Add-Ins) и щелкните на кнопке **Перейти** (Go). Для поиска файла надстройки щелкните на кнопке **Обзор** (Browse). После установки надстройка будет доступной между сеансами Excel. Надстройка также будет работать, если она открыта с помощью команды **Файл⇨Открыть** (File⇨Open).

Создание мастера расчета займа

Приложение мастера расчета займа, которое в свое время начиналось с простой концепции, стало достаточно сложным проектом. Основной его целью была демонстрация максимального количества методов разработки приложений. При этом нужно было разработать программу, от которой была бы реальная отдача. Конечный результат можно было увидеть еще до начала разработки приложения, но основной целью было создание конечного приложения, а не получение расчетных данных.

Главная цель приложения традиционно очень проста. Необходимо создать приложение, которое собирает от пользователя информацию и создает рабочий лист. Но в процессе разработки возникли идеи по расширению возможностей приложения. В результате рабочий процесс несколько раз заходил в тупик. Кто-то может назвать эти эксперименты напрасной тратой времени, но они являются неотъемлемой частью процесса разработки приложения.

Разработка проекта была завершена в течение одного дня. Еще несколько часов было потрачено на доводку и тестирование приложения. В версию приложения, включенного в это издание книги, были добавлены дополнительные улучшения.

Изменение пользовательского интерфейса

Для созданной надстройки следует определить способ доступа. Для этого используется простой код RibbonX, с помощью которого добавляется кнопка в новую группу на вкладке **Вставка** (Insert), как показано на рис. 25.7. После щелчка на этой кнопке вызывается процедура `StartAmortizationWizard`, которая отображает пользовательское диалоговое окно FormMain.

Ниже представлен код RibbonX, который создает кнопку на ленте.

```
<customUI xmlns="http://schemas.microsoft.com/office/2006/ _
   01/customui">
   <ribbon>
      <tabs>
         <tab idMso="TabInsert">
            <group id="gpUtils" label="Loan Amortization">
```

```
                <button id="b1"
                size="large"
                imageMso="CreateQueryFromWizard"
                label="Loan Amortization Wizard"
                supertip="Щелкните для создания графика _
                   погашения займа."
                onAction="StartAmortizationWizard"/>
             </group>
          </tab>
       </tabs>
   </ribbon>
</customUI>
```

Перекрестная ссылка

Обратитесь к главе 22 для получения сведений о модификации ленты.

Отображение начального сообщения

На протяжении ряда лет я инсталлировал множество надстроек Excel и пришел к выводу о том, что большинство из них "недружественны" к пользователю. Для повышения степени "дружественности" приложения было добавлено пользовательское диалоговое окно, которое отображается при открытии рабочей книги. В этом окне появляется подсказка, которая сообщает о способе запуска мастера на выполнение. Это окно показано на рис. 25.8.

Рис. 25.7. Новая группа вкладки Вставка содержит один элемент управления

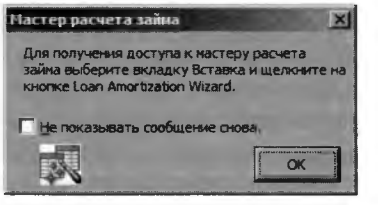


Рис. 25.8. Эта форма отображается после открытия мастера расчета займа

В пользовательском диалоговом окне содержится опция, с помощью которой можно отключить отображение подсказки.

Ниже представлен код процедуры `Workbook_Open`, отображающей пользовательское диалоговое окно.

```
Private Sub Workbook_Open()
   If GetSetting(APPNAME, "Defaults", "ShowMessage", _
      "Yes") = "Yes" Then
      FormMessage.Show
   End If
End Sub
```

Выбранный пользователем вариант отображения пользовательского диалогового окна хранится в системном реестре Windows. Ключ системного реестра определяется именем приложения (глобальная константа `APPNAME`). Значение, заданное по умолчанию, — "Yes" (Да), поэтому пользовательское диалоговое окно будет отображаться как минимум один раз.

Ниже приводится код, вызываемый после щелчка пользователем на кнопке OK.

```
Private Sub OKButton_Click()
   If cbMessage Then
      SaveSetting APPNAME, "Defaults", "ShowMessage", "No"
   Else
      SaveSetting APPNAME, "Defaults", "ShowMessage", "Yes"
   End If
   Unload Me
End Sub
```

Если в пользовательском диалоговом окне устанавливается соответствующий флажок, настройка реестра получит значение "No" (Нет), и пользовательское диалоговое окно не будет отображаться повторно.

Инициализация диалогового окна `FormMain`

Процедура `UserForm_Initialize` для диалогового окна `FormMain` выполняет достаточно большой объем работы.

- Значение свойства `Style` элемента управления `MultiPage` устанавливается равным `fmTabStyleNone`. Вкладки, представленные в окне Visual Basic Editor, облегчают редактирование пользовательского диалогового окна.
- Значение свойства `Value` элемента управления `MultiPage` устанавливается равным `0`. Это обеспечивает отображение первой страницы элемента управления `MultiPage`, независимо от того, когда рабочая книга сохранялась в последний раз.
- Добавляются опции к раскрывающимся спискам трех элементов управления `ComboBox`, которые находятся в диалоговом окне.
- Вызывается процедура `GetDefaults` для установки значений, которые использовались в последний раз и хранятся в системном реестре (информация об этом приведена в разделе "Сохранение и получение значений по умолчанию").
- Проверяется активность рабочей книги. Если книга неактивна, то отключается элемент управления `OptionButton`, который позволяет пользователю создать новый рабочий лист в активной рабочей книге.
- Если рабочая книга активна, то в результате дополнительной проверки определяется защищенность структуры рабочей книги. Если это так, то процедура отключает элемент управления `OptionButton`, который позволяет создавать новый рабочий лист в активной рабочей книге.

Обработка событий в процессе отображения диалогового окна `UserForm`

Модуль кода для диалогового окна `FormMain` содержит несколько процедур обработки событий `Click` и `Change` для элементов управления, которые находятся в диалоговом окне `UserForm`.

Перекрестная ссылка

Щелчок на кнопках Назад и Далее определяет, какая из страниц элемента управления `MultiPage` будет отображена следующей. Процедура `MultiPage1_Change` изменяет заголовок диалогового окна `UserForm`, а также включает/отключает кнопки Назад и Далее в зависимости от того, какая страница элемента управления отображается в данный момент. Дополнительные сведения о программировании мастеров можно найти в главе 15.

Отображение справочной информации

Для отображения справочной информации существует несколько способов. В данном случае используется простая методика, в которой применяется диалоговое окно `UserForm`. Это диалоговое окно отображает текст, который хранится на рабочем листе (рис. 25.9). Можно заметить, что предоставляемая справочная информация зависит от контекста. Когда пользователь щелкает на кнопке Справка, отображается раздел справочного руководства, который относится к активной странице элемента управления `MultiPage`.

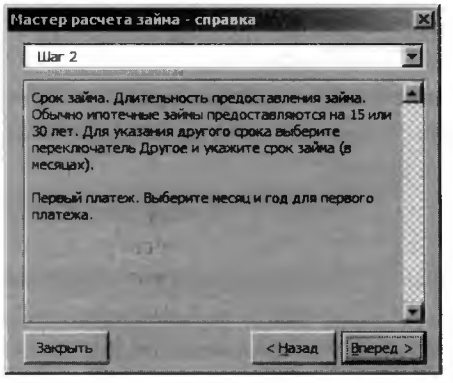


Рис. 25.9. Справочные сведения представлены в пользовательском диалоговом окне, куда копируется текст из рабочего листа

Рабочие листы надстройки обычно невидимы. Для просмотра листов надстройки, включающих текст справки, следует свойству рабочей книги `IsAddin` временно присвоить значение `False`. Один из методов, применяемых в данном случае, — выбрать проект в окне Project и в окне отладки (Immediate) выполнить следующий оператор:

```
ThisWorkbook.IsAddin = False
```

Перекрестная ссылка

Для получения дополнительных сведений о передаче текста из рабочего листа в пользовательское диалоговое окно обратитесь к главе 24.

Создание нового рабочего листа

Щелчок на кнопке Готово приведет к выполнению основной задачи приложения. Процедура обработки события `Click` для этой кнопки выполняет следующие действия.

1. Вызывается функция `DataIsValid`, которая проверяет введенные пользователем данные на правильность. Если все введенные данные корректны, то функция возвращает значение `True` и процедура продолжает свое выполнение. Если обнаружены некорректные входные данные, функция `DataIsValid` активизирует элемент управления, который содержит неверные данные, и создает окно сообщения с описанием проблемы (рис. 25.10).

2. Если введенные пользователем данные корректны, то процедура создает новый рабочий лист в активной рабочей книге или в новой рабочей книге, что определяется выбором пользователя.

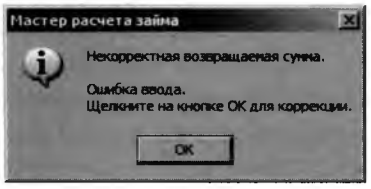


Рис. 25.10. Если пользователь ввел некорректные данные, активизируется элемент управления, содержащий ошибку

3. *Параметры* кредитования (возвращаемая сумма, первый взнос, сумма займа, срок погашения и процентная ставка) записываются на этом рабочем листе. Здесь не обойтись без операторов `If`, так как сумма первого взноса может указываться и в виде процента от возвращаемой суммы, и в виде фиксированного значения.
4. На рабочий лист заносятся заголовки столбцов.
5. Первая строка формул вводится под строкой заголовков. Первая строка отличается от остальных, так как формулы в ней ссылаются на данные, которые хранятся в области данных о кредите. Остальные формулы ссылаются на значения в предыдущей строке. Обратите внимание на то, что в формулах используются именованные диапазоны. Эти имена определены на уровне листа, поэтому пользователь может хранить более одного графика погашения займа в одной и той же рабочей книге.
6. Чтобы задать неименованные ссылки, используется стиль R1C1 (что намного проще, чем определение фактического адреса ячейки).
7. Вторая строка формул вводится на рабочий лист и копируется в нижние строки для каждого месяца.
8. Если пользователь затребовал отображение ежегодных, а не ежемесячных данных, то в процедуре запускается метод `Subtotal` для расчета необходимых значений. Этот подход демонстрирует гибкость встроенных средств Excel, которые помогают избежать написания *дополнительного* кода.
9. По причине того что создание сумм в столбце `Баланс` теперь не имеет смысла, процедура заменяет формулу в столбце `Баланс` на формулу, которая возвращает значение баланса на ежегодной основе.
10. Когда Excel подсчитывает суммы, она выделяет их с помощью границ. Если пользователь не требовал выделять суммы границами, то процедура использует метод `ClearOutline` для их отмены.
11. После этого процедура производит форматирование информации, которая отображается в ячейках: в данном случае изменяется формат отображения чисел, а также применяется средство **Автоформат**, если пользователь требует выделять ячейки цветом.
12. Затем план погашения займа преобразуется в таблицу и также применяется стиль, основанный на выбранном пользователем цвете или оттенках серого.

13. Далее процедура меняет ширину столбцов, блокирует заголовки и защищает формулы, а также несколько ключевых ячеек, значения в которых нельзя редактировать. После этого рабочий лист считается защищенным, хотя пароль не используется.
14. Если при выполнении пятого пункта этого списка установлен флажок **Защита листа** (Protect Sheet), лист будет защищен (но без применения пароля).
15. Наконец, процедура `SaveDefaults` сохраняет сведения о состоянии элементов управления диалогового окна `UserForm` в системном реестре. Эти значения будут использоваться по умолчанию в следующий раз, когда пользователь примется за создание расписания погашения займа (дополнительная информация приведена в следующем разделе).

Сохранение и получение значений по умолчанию

Если запустить это приложение, то можно заметить, что диалоговое окно `UserForm`, в котором определяются параметры кредита (`FormMain`), всегда отображает значения, которые использовались в последний раз. Другими словами, диалоговое окно "запоминает" последние данные и применяет их в качестве новых значений по умолчанию. Таким образом упрощается создание нескольких сценариев "что-если" для расчета погашения кредита, которые отличаются только одним параметром. Подобный эффект достигается благодаря хранению значений в системном реестре и получению этих данных на этапе инициализации диалогового окна `UserForm`. Когда приложение запускается в первый раз, системный реестр не содержит значений, поэтому используются значения, принятые по умолчанию в элементах управления диалогового окна `UserForm`.

Следующая процедура, которая называется `GetDefaults`, циклически просматривает элементы управления в диалоговом окне `UserForm`. Если элемент управления имеет тип `TextBox`, `ComboBox`, `OptionButton`, `CheckBox` или `SpinButton`, то процедура вызывает функцию VBA `GetSetting` и считывает значения из системного реестра. Обратите внимание на то, что третий аргумент функции `GetSetting` представляет значение, которое используется в том случае, если в системном реестре не найдено требуемых параметров. В результате применяется значение, которое указывается для элемента управления на этапе проектирования. `APPNAME` является глобальной константой, содержащей название приложения.

```
Sub GetDefaults()
'  Чтение заданных по умолчанию настроек из реестра
   Dim ctl As Control
   Dim CtrlType As String

   For Each ctl In Me.Controls
      CtrlType = TypeName(ctl)
      If CtrlType = "TextBox" Or _
         CtrlType = "ComboBox" Or _
         CtrlType = "OptionButton" Or _
         CtrlType = "CheckBox" Or _
         CtrlType = "SpinButton" Then
         ctl.Value = GetSetting _
            (APPNAME, "Defaults", ctl.Name, ctl.Value)
      End If
   Next ctl
End Sub
```

На рис. 25.11 показаны эти значения в реестре, просматриваемом с помощью редактора реестра Windows (Windows Registry Editor).

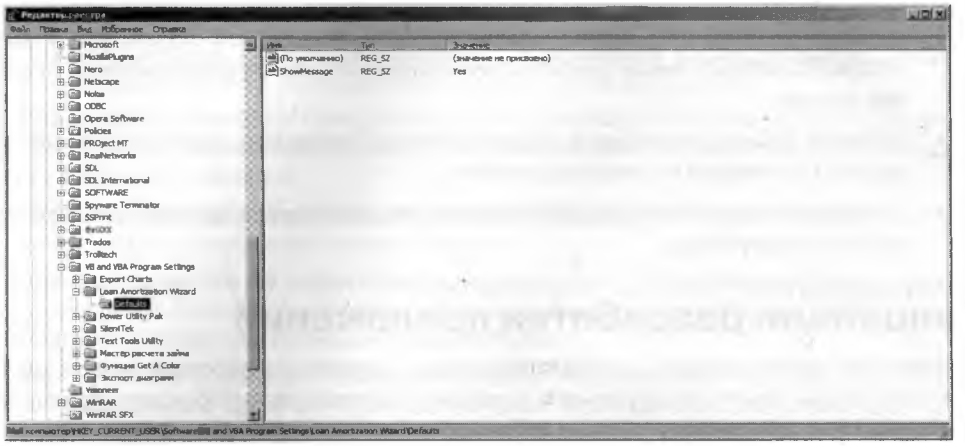


Рис. 25.11. В системном реестре Windows хранятся значения, заданные по умолчанию для мастера

Ниже приведен код процедуры `SaveDefaults`, подобной процедуре `GetDefaults`. При этом используется оператор VBA `SaveSetting` для записи текущих значений в системный реестр.

```
Sub SaveDefaults()
'  Запись текущих настроек в системный реестр
   Dim ctl As Control
   Dim CtrlType As String
   For Each ctl In Me.Controls
      CtrlType = TypeName(ctl)
      If CtrlType = "TextBox" Or _
         CtrlType = "ComboBox" Or _
         CtrlType = "OptionButton" Or _
         CtrlType = "CheckBox" Or _
         CtrlType = "SpinButton" Then
         SaveSetting APPNAME, "Defaults", ctl.Name, _
            CStr(ctl.Value)
      End If
   Next ctl
End Sub
```

Обратите внимание на то, что в коде используется функция `CStr`, которая преобразует каждую настройку в строку, — тем самым предотвращаются неполадки у тех пользователей, которые установили другие региональные стандарты.

И функция `GetSetting`, и оператор `SaveSetting` всегда используют приведенный ниже раздел системного реестра для хранения значений.

```
HKEY_CURRENT_USER\Software\VB and VBA Program Settings\
```

Возможные улучшения

Считается, что создание приложения никогда не завершается — просто прекращается работа над внесением улучшений. Даже затратив немного времени, можно придумать несколько улучшений, которые можно внести в приложение мастера расчета займа:

- добавить элемент управления, который позволит пользователю ознакомиться с накопительными итогами, характеризующими состояние погашения процента и общей суммы кредита;
- добавить возможность использования дробной процентной ставки, что позволит создавать долгосрочные прогнозы, рассчитанные для разных значений процентной ставки;
- добавить больше параметров форматирования (например, отключить десятичные разряды и отобразить символы доллара);
- предоставить пользователю возможность указывать текст верхнего и нижнего колонтитулов страницы.

Концепции разработки приложений

Зачастую сложно понять логику приложения, которое разработано кем-то другим. Для того чтобы помочь разобраться с основами создания рассмотренного приложения, в исходный код приложения обычно добавляются комментарии. Но если вам важно понять принципы, лежащие в основе определенного приложения, то воспользуйтесь отладчиком для пошагового выполнения исходного кода.

Приложение мастера расчета займа в полной мере демонстрирует основные методы и концепции разработки приложений в Excel:

- изменение ленты;
- использование пользовательского диалогового окна (в виде мастера) для сбора информации;
- динамическая настройка свойства `Enabled` элемента управления;
- связывание элементов управления `TextBox` и `SpinButton`;
- отображение пользовательской справки;
- именование ячеек с помощью VBA;
- запись и копирование формул с помощью VBA;
- чтение и запись значений в системном реестре.

Полезный список вопросов

При разработке приложения, ориентированного на пользователя, следует помнить о некоторых ключевых моментах. Следующий список будет служить вам полезным напоминанием.

- **Поддерживают ли диалоговые окна работу с помощью клавиатуры?** Не забудьте добавить комбинации клавиш, а также проверьте правильность порядка перехода между элементами управления.
- **Делает ли приложение предположения о существовании папок?** Если приложение читает или записывает данные в файлы, то следует ожидать, что папка по умолчанию уже существует.
- **Было ли разработано средство для закрытия всех диалоговых окон?** Вряд ли можно ожидать, что пользователь будет закрывать диалоговое окно щелчком на кнопке OK, но это вполне реально.

- **Делается ли предположение о том, что больше не открыт ни один рабочий лист?** Если приложение является единственной рабочей книгой, которая открыта во время тестирования, то может оказаться, что не рассмотрены случаи, когда открыто несколько рабочих книг.
- **Осуществлялось ли предположение о видимости рабочей книги?** Работать с Excel можно и тогда, когда не открыта ни одна рабочая книга.
- **Делались ли попытки оптимизации приложения?** Например, можно увеличить скорость выполнения приложения, если явно объявить переменные, а также задать объектные переменные.
- **Является ли адекватной документация к процедуре?** Можно ли понять код, если вернуться к нему через шесть месяцев?
- **Предоставляется ли адекватная документация для конечного пользователя?** Выполнение этого условия сокращает (или даже полностью исключает) количество вопросов со стороны конечных пользователей.
- **Выделено ли время для проверки работы приложения?** Приложение не может быть идеальным сразу же после разработки. Поэтому нужно выделить время для его улучшения.

Разработка ориентированных на пользователя приложений в Excel — непростая задача. Вы должны быть полностью осведомлены о том, каким образом используется ваше приложение. И хотя я пытался сделать приложение сверхустойчивым, я не выполнял интенсивное тестирование на основе реальных примеров, поэтому не удивлюсь, если при определенных условиях приложение не будет работать.

Часть VII

Дополнительные темы

В этой части...

Вопросы совместимости

В этой главе...

- Концепция совместимости
- Проблемы совместимости
- Избегайте использования новых возможностей
- Поддержка платформы Macintosh
- Использование 64-разрядной версии Excel
- Создание интернациональных приложений

Если созданное вами приложение предназначено для использования с более ранними версиями Excel, версиями для Macintosh или версиями с другими локальными настройками языка, следует учесть ряд вопросов, которые являются предметом рассмотрения этой главы.

Концепция совместимости

Совместимость — это часто используемый термин в среде разработчиков компьютерных приложений. Данный термин обозначает правильность выполнения приложения в различных условиях. Эти условия определяются настройками аппаратного или программного обеспечения, а также принципами их взаимодействия. Например, программное обеспечение, которое создано специально для Windows, не будет работать под управлением таких операционных систем, как MacOS или Linux.

В этой главе рассматриваются вопросы совместимости, а именно: особенности работы приложений Excel 2010 с более ранними версиями Excel для Windows и Excel для Macintosh. Даже если две версии Excel используют один и тот же формат, это вовсе не гарантирует полной совместимости между создаваемыми ими файлами. Например, программы Excel 97, Excel 2000, Excel 2002, Excel 2003 и Excel 2002 для Macintosh используют похожий файловый формат, но проблемы совместимости все равно остаются.

И еще один пример. В Excel 2010 и Excel 2007 применяется идентичный файловый формат. Но если ваше приложение использует свойства, которые появились в Excel 2010, вы не вправе ожидать, что пользователи Excel 2007 магическим образом получат доступ к этим новым свойствам.

Итак, Excel постоянно изменяется, совершенствуется и обновляется, поэтому не существует способа обеспечить полную совместимость создаваемых приложений. К сожалению, автоматически обеспечить совместимость приложения с различными версиями невозможно. В большинстве случаев необходимо выполнить достаточно большой объем дополнительной работы, чтобы достигнуть приемлемого уровня совместимости.

Проблемы совместимости

Следует помнить о нескольких типах проблем совместимости.

- **Несовместимость форматов файлов.** Рабочие книги могут сохраняться в различных файловых форматах Excel. Более ранние версии Excel не могут открывать рабочие книги, сохраненные с применением более позднего файлового формата. Дополнительные сведения о работе с файлами Excel 2010 (и Excel 2007) в более ранних версиях Excel можно найти во врезке "Microsoft Office Compatibility Pack".
- **Несовместимость новых средств.** Очевидно, что новые возможности, которые появились в поздних версиях Excel, не могут использоваться в старых версиях этой программы.
- **Проблемы, связанные с компанией Microsoft.** Компания Microsoft иногда сама вызывает проблемы несовместимости. Например, как отмечалось в главе 23, номера индексов для контекстных меню несовместимы в различных версиях Excel.
- **Совместимость Windows и Macintosh.** Если приложение должно работать под управлением обеих платформ, то придется потратить намного больше времени для решения всех проблем совместимости.
- **Проблемы, связанные с разрядностью.** Программа Excel 2010 — это первая версия Excel, которая доступна в виде 32- и 64-разрядной версий. Если код VBA использует API-функции, следует знать о некоторых потенциальных проблемах, возникающих в случае выполнения кода в 32- и 64-разрядной версии Excel либо одновременно в двух версиях Excel.
- **Вопросы языковой совместимости.** Если приложение будет использоваться в иноязычных версиях Excel, то обратите внимание на целый ряд дополнительных вопросов языковой совместимости.

После изучения этой главы вам должно стать ясно, что существует один гарантированный способ достижения совместимости: приложение необходимо протестировать на всех целевых платформах, а также во всех интересующих версиях Excel. Зачастую это просто невозможно. Однако существуют некоторые способы, которые помогают разработчику частично гарантировать работу приложения в различных версиях Excel.

Примечание

В этой главе вы не найдете списка всех проблем несовместимости различных версий Excel — подобного списка просто не существует, а создать его практически невозможно.

Совет

Хорошим источником информации о потенциальных проблемах совместимости является интерактивная база знаний Microsoft, которая находится по адресу `http://support.microsoft.com`. Эта база позволит идентифицировать проблему, которая возникает в определенной версии Excel.

Избегайте использования новых возможностей

Если приложение должно работать в нескольких версиях Excel (Excel 2010 и более ранних), то следует избегать использования новых возможностей, добавленных после выхода самой ранней версии Excel, которую необходимо поддерживать. Еще одной альтернативой является выборочное использование новых возможностей. Другими словами, приложение должно определить, какая версия Excel применяется, и на основании этого принимать решение об использовании нового средства.

Разрабатывая приложения с помощью VBA, не следует использовать объекты, методы и свойства, которые не доступны в более ранних версиях. Самым безопасным подходом является разработка приложения на основе "наибольшего общего знаменателя". Чтобы обеспечить совместимость с Excel 2000 и более поздними версиями, необходимо для разработки приложения использовать Excel 2000, после чего продукт должен быть всесторонне протестирован в остальных средах.

Microsoft Office Compatibility Pack

Если вы планируете разрабатывать приложения в среде Excel 2010, используя также более ранние версии этой программы, можете поступить двояко:

- сохранить файлы в старом файловом формате XLS;
- передать всем пользователям, просматривающим ваши файлы, пакет Microsoft Office Compatibility Pack.

Программу Microsoft Office Compatibility Pack можно загрузить с веб-сайта Microsoft по адресу `www.microsoft.com`. После установки этой программы пользователи пакетов Office XP и Office 2003 могут открывать, редактировать и сохранять документы, рабочие книги и презентации в новых файловых форматах для Word, Excel и PowerPoint.

Обратите внимание: пакет Microsoft Office Compatibility Pack не добавляет в прежние версии Excel новые свойства, присущие Excel 2007 либо Excel 2010. Она просто позволяет открывать и сохранять файлы в новом формате.

Еще в версии Excel 2007 появилось новое средство проверки совместимости, которое получило дальнейшее развитие в версии Excel 2010 (рис. 26.1). Для запуска этой программы выберите команду **Файл⇨Сведения⇨Поиск проблем⇨Проверка совместимости** (File⇨Info⇨Check For Issues⇨Check Compatibility). Благодаря этой программе идентифицируются все возможные проблемы совместимости, возникающие при открытии файла в более ранних версиях Excel.

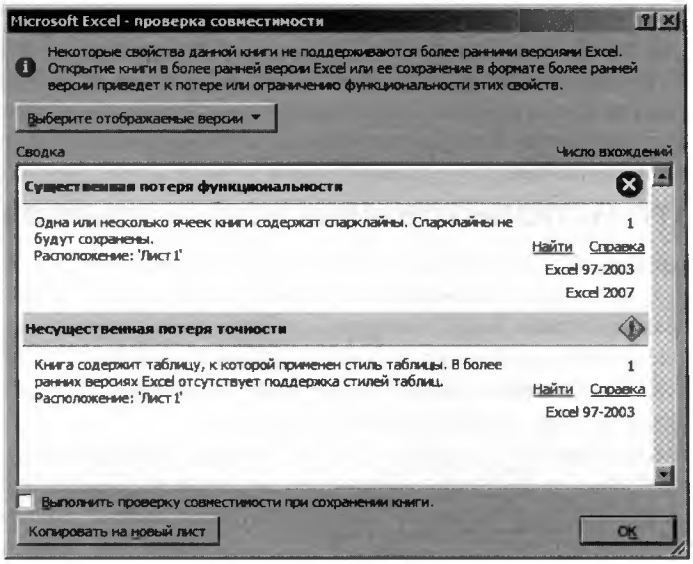


Рис. 26.1. Средство проверки совместимости

К сожалению, программа проверки совместимости даже не "замечает" код VBA — именно он и порождает проблемы совместимости. Эту проблему можно решить путем загрузки приложения Microsoft Office Code Compatibility Inspector (с веб-сайта `http://Microsoft.com`). Это приложение оформлено в виде надстройки и после установки доступно на вкладке Разработчик (Developer). Именно эта программа поможет вам найти потенциально проблемные фрагменты кода VBA.

Поддержка платформы Macintosh

Чаще всего проблемы совместимости касаются компьютеров Macintosh. По причине того что Excel для Macintosh составляет небольшую часть рынка Excel, многие разработчики просто игнорируют эту платформу. Хорошая новость состоит в том, что старый файловый формат XLS совместим с обеими платформами. Плохая новость заключается в том, что наборы поддерживаемых свойств неодинаковы, а совместимость на уровне макросов VBA далека от идеальной.

Определение номера версии Excel

Свойство `Version` объекта `Application` возвращает номер версии Excel. Возвращаемое значение представляет собой строку, которую можно преобразовать в числовое значение. Для этого используется функция VBA под названием `Val`. Приведенная ниже функция возвращает значение `True`, если выполняется Excel 2007 или более поздняя версия этой программы.

```
Function XL12OrLater()
   XL12OrLater = Val(Application.Version) >= 12
End Function
```

Обратите внимание: номер версии Excel 2007 — 12, а Excel 2010 — 14. По вполне понятным причинам версия с номером 13 отсутствует.

Примечание

Далее предполагается, что вы используете устаревшие версии Excel для Macintosh, которые поддерживают VBA.

Можно создать код VBA, который будет определять, на какой платформе работает приложение. Следующая функция получает доступ к свойству `OperatingSystem` объекта `Application` и возвращает значение `True`, если используется одна из версий Windows (т.е. если возвращаемая строка содержит значение "Win").

```
Function WindowsOS() As Boolean
   If Application.OperatingSystem like "*Win*" Then
      WindowsOS = True
   Else
      WindowsOS = False
   End If
End Function
```

Между Windows- и Mac-версией Excel существует ряд отличий. Некоторые из них можно считать "косметическими" (например, по умолчанию используются разные системные шрифты). Но существуют и более серьезные проблемы. Например, в состав Excel для Macintosh не входят компоненты ActiveX. Кроме того, Excel для Macintosh использует систему дат "1904", поэтому рабочие книги, в которых применяется система дат по умолчанию, могут опережать время на 4 года. Excel для Windows по умолчанию использует систему дат "1900". Это означает, что в Macintosh дата 1 будет означать 1 января 1904 года. В то же время в Windows такая дата будет означать 1 января 1900 года.

Еще одно ограничение касается вызова функций Windows API — в Excel для Macintosh они не работают. Если приложение полностью зависит от таких функций, то вам придется разрабатывать дополнительные способы выполнения заданий, возложенных на приложение.

Если код работает с именами файлов, значит, необходимо указать путь с использованием подходящего разделителя (двоеточие в Macintosh и обратная косая черта в Windows). Правильнее всего динамически определить подходящий разделитель в пути с помощью кода VBA. Представленный ниже оператор назначает символ разделителя пути переменной `PathSep`.

```
PathSep = Application.PathSeparator
```

После выполнения этого оператора код может использовать переменную `PathSep` для замещения строго определенного разделителя.

Вместо того чтобы создавать один файл, совместимый с обеими платформами, многие разработчики принимают решение о сохранении двух версий приложения для каждой платформы. Сначала создается приложение для одной платформы (обычно это Excel для Windows), после чего оно модифицируется для работы на другой платформе. Другими словами, вам придется отлаживать две различные версии приложения.

Существует только один способ обеспечения совместимости приложения с Excel для Macintosh: приложение должно быть всесторонне протестировано на этой платформе. Будьте готовы к разработке обходных путей, которые заменят отсутствующие или неработающие возможности.

Использование 64-разрядной версии Excel

При установке Excel 2010 можно выбрать 32- либо 64-разрядную версию приложения. Последняя версия может устанавливаться и работать только на 64-разрядной версии

Windows. В 64-разрядной версии Excel могут обрабатываться огромные рабочие книги, поскольку в этом случае задействуется потенциал пространства адресов 64-разрядной версии Windows.

Большинство пользователей не нуждаются в 64-разрядной версии Excel, поскольку они не нуждаются в обработке больших объемов данных в рабочих книгах. К тому же 64-разрядная версия Excel не обеспечивает прироста производительности. Более того, некоторые операции в 64-рязрядной версии Excel выполняются даже медленнее, чем в 32-разрядной версии.

Как правило, рабочие книги, созданные в 32-разрядной версии Excel, могут открываться и обрабатываться в 64-разрядной версии Excel. Единственная проблема может возникать в тех случаях, если рабочая книга включает код VBA, использующий функции Windows API. Причина возникновения этой проблемы заключается в том, что объявления 32-разрядных API-функций не могут компилироваться в 64-разрядной версии Excel.

Например, следующее объявление хорошо работает в 32-разрядных версиях Excel, но в случае его выполнения в 64-разрядной версии Excel 2010 возникает ошибка компиляции.

```
Declare Function GetWindowsDirectoryA Lib "kernel32" _
   (ByVal lpBuffer As String, ByVal nSize As Long) As Long
```

Следующее объявление может использоваться в Excel 2010 (как в 32-, так и в 64-разрядной), но в случае использования в предыдущих версиях Excel приводит к появлению ошибки компиляции.

```
Declare PtrSafe Function GetWindowsDirectoryA Lib "kernel32" _
   (ByVal lpBuffer As String, ByVal nSize As Long) As Long
```

Для использования этой API-функции в 32- и 64-разрядной версиях Excel следует объявить две ее версии с помощью следующих условных директив компилятора.

- Директива `VBA7` возвращает значение `True`, если код использует VBA версии 7 (эта версия включена в состав Office 2010).
- Директива `Win64` возвращает значение `True`, если код выполняется под управлением 64-разрядной версии Excel.

Примечание

В операционной системе можно установить одну версию VBA. Поэтому, если наравне с устаревшими версиями Excel устанавливается Excel 2010, а затем активизировать приложение VB Editor в одной из устаревших версий Excel и выполнить команду Help⇨About Microsoft Visual Basic (Справка⇨ О программе Microsoft Visual Basic), появится диалоговое окно с сообщением об отсутствии поддержки VBA 7.

Ниже приводится пример использования этих директив для объявления API-функций, совместимых с 32- и 64-разрядной версиями Excel.

```
#If VBA7 And Win64 Then
   Declare PtrSafe Function GetWindowsDirectoryA Lib "kernel32" _
      (ByVal lpBuffer As String, ByVal nSize As Long) As Long
#Else
   Declare Function GetWindowsDirectoryA Lib "kernel32" _
   (ByVal lpBuffer As String, ByVal nSize As Long) As Long
#End If
```

Первый оператор `Declare` применяется только в том случае, если обе директивы (`VBA7` и `Wind64`) возвращают значение `True` — это наблюдается только в случае ис-

пользования 16-разрядной версии Excel 2010. Для всех остальных версий Excel применяется второй оператор `Declare`.

Создание интернациональных приложений

Последний вопрос совместимости касается языковых и интернациональных настроек. Excel поставляется в различных языковых версиях. Следующий оператор выводит значение кода страны текущей версии Excel:

```
MsgBox Application.International(xlCountryCode)
```

Версия Excel, предназначенная для применения в Великобритании и США, имеет код 1. Остальные коды языков перечислены в табл. 26.1.

Таблица 26.1. Коды языков, поддерживаемые в Excel

Язык	Код
Английский	1
Русский	7
Греческий	30
Голландский	31
Французский	33
Испанский	34
Венгерский	36
Итальянский	39
Чешский	42
Датский	45
Шведский	46
Норвежский	47
Польский	48
Немецкий	49
Португальский (бразильский)	55
Тайский	66
Японский	81
Корейский	82
Вьетнамский	84
Упрощенный китайский	86
Турецкий	90
Хинди	91
Урду	92
Португальский	351
Финский	358
Традиционный китайский	886
Арабский	966
Иврит	972
Фарси	982

В Excel также поддерживаются языковые пакеты, в результате чего единственная копия Excel может "говорить" с пользователями на различных языках. Причем языки вступают в игру в двух областях — пользовательский интерфейс и режим выполнения.

Для определения языка, используемого интерфейсом пользователя, применяется следующий оператор:

```
Msgbox Application.LanguageSettings.LanguageID(msoLanguageIDUI)
```

Например, идентификатор английского языка — 1033.

Если ваше приложение предназначено для пользователей, говорящих на иностранном языке, следует убедиться в том, что в выбран корректный язык. Также следует указать символы разделителя десятичной точки и разделителя тысяч, применяемые в каждом конкретном случае. В США в этих целях практически всегда применяются точка и запятая соответственно. Пользователи из других стран могут использовать собственные символы разделителей. Также следует обратить внимание на формат даты и времени. Например, в США применяется не вполне логичный формат месяц/день/год.

Если разрабатывается приложение, которое будет применяться только пользователями одной компании, то о вопросах интернациональной совместимости можно не задумываться. Если же офисы компании рассредоточены по всему миру или планируется распространение приложения за пределы страны, то стоит обратить внимание на ряд вопросов, решение которых обеспечит правильную работу приложения. Эти вопросы рассматриваются в данном разделе.

Многоязычные приложения

Весьма очевидным моментом является язык общения, который будет использоваться в приложении. Например, если приложение отображает одно или несколько диалоговых окон, то может возникнуть необходимость в отображении текста на языке пользователя. Это несложно (если, конечно, вы владеете соответствующим языком).

Компакт-диск

На прилагаемом к книге компакт-диске в файле `multilingual wizard.xlsm` находится пример, демонстрирующий выбор одного из трех языков в диалоговом окне: английский, испанский либо немецкий.

Первый этап работы мастера заключается в отображении элемента управления `OptionButton`, позволяющего пользователю указать язык общения. Текст на трех языках хранится на рабочем листе.

Процедура `UserForm_Initialize` содержит код, который пытается "угадать" используемый пользователем язык путем проверки значения свойства `International`.

```
Select Case Application.International(xlCountryCode)
   Case 34 'испанский
      UserLanguage = 2
   Case 49 'немецкий
      UserLanguage = 3
   Case Else 'по умолчанию английский
      UserLanguage = 1 'по умолчанию
End Select
```

На рис. 26.2 показано пользовательское диалоговое окно, отображающее текст на трех языках.

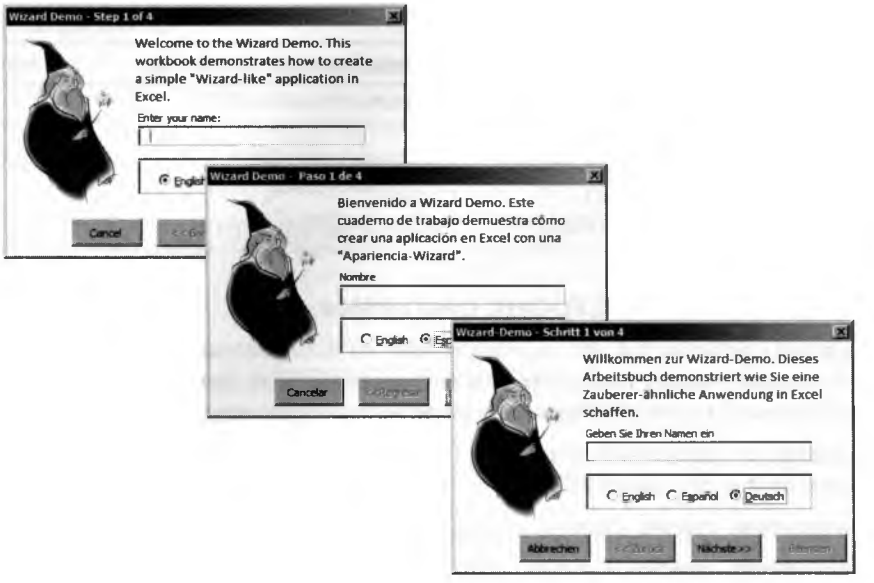


Рис. 26.2. Демонстрация работы мастера на английском, испанском и немецком языках

Язык в VBA

Как правило, задумываться о языке при создании кода VBA не стоит. Excel использует две библиотеки объектов: Excel и VBA. При установке Excel регистрируется соответствующая версия этих библиотек. Данные версии применяются по умолчанию (они не зависят от языковой версии Excel).

Использование "локальных" свойств

Если код будет отображать информацию рабочего листа, например диапазон адресов, то, скорее всего, необходимо использовать язык общения с пользователем. Например, следующий оператор отображает формулу в ячейке A1:

```
MsgBox Range("A1").Formula
```

Для интернациональных приложений наиболее удачным подходом будет использование свойства `FormulaLocal` вместо свойства `Formula`.

```
MsgBox Range("A1").FormulaLocal
```

"Локальные" версии свойств предоставляют еще несколько возможностей. Эти свойства показаны в табл. 26.2 (дополнительную информацию можно получить, обратившись к справочному руководству).

Таблица 26.2. "Локальные" версии свойств

Свойство	"Локальная" версия	Возвращаемое значение
`Address`	`AddressLocal`	Адрес
`Category`	`CategoryLocal`	Категория функции (только для макросов XLM)
`Formula`	`FormulaLocal`	Формула

Окончание табл. 26.2

Свойство	"Локальная" версия	Возвращаемое значение
`FormulaR1C1`	`FormulaR1C1Local`	Формула в записи R1C1
`Name`	`NameLocal`	Имя
`NumberFormat`	`NumberFormatLocal`	Числовой формат
`RefersTo`	`RefersToLocal`	Ссылка
`RefersToR1C1`	`RefersToR1C1Local`	Ссылка в записи R1C1

Идентификация настроек системы

Очень редко случается так, что система конечного пользователя настроена так же, как и система, в которой происходила разработка приложения. Для интернациональных приложений необходимо получить информацию о следующих параметрах настройки системы.

- **Разделитель десятичных знаков.** Символ, который используется для отделения десятичной части значения.
- **Разделитель тысяч.** Символ, который используется для разделения триад цифр.
- **Разделитель списка.** Символ, который используется для разделения элементов списка.

Текущее значение параметра настройки можно получить с помощью свойства `International` объекта `Application`. Например, следующий оператор отображает разделитель десятичных знаков, который не всегда является запятой:

```
MsgBox Application.International(xlDecimalSeparator)
```

Существует 45 интернациональных параметров настройки, доступ к которым можно получить с помощью свойства `International`. Эти параметры перечислены в табл. 26.3.

Таблица 26.3. Константы свойства `International`

Константа	Что определяет
`xlCountryCode`	Языковую версию Microsoft Excel
`xlCountrySetting`	Текущие установки страны в папке Панель управления
`xlDecimalSeparator`	Разделитель десятичной части
`xlThousandsSeparator`	Разделитель нулей или триад цифр
`xlListSeparator`	Разделитель списка
`xlUpperCaseRowLetter`	Символ строки в верхнем регистре (для ссылок в стиле R1C1)
`xlUpperCaseColumnLetter`	Символ столбца в верхнем регистре
`xlLowerCaseRowLetter`	Символ строки в нижнем регистре
`xlLowerCaseColumnLetter`	Символ столбца в нижнем регистре
`xlLeftBracket`	Символ, который используется вместо открывающей квадратной скобки (`[`) в относительных ссылках стиля R1C1
`xlRightBracket`	Символ, который используется вместо закрывающей квадратной скобки (`]`) в относительных ссылках стиля R1C1
`xlLeftBrace`	Символ, который используется вместо открывающей фигурной скобки (`{`) в массивах символов
`xlRightBrace`	Символ, который используется вместо закрывающей фигурной скобки (`}`) в массивах символов

Продолжение табл. 26.3

Константа	Что определяет
`xlColumnSeparator`	Символ, который используется для разделения столбцов в массивах символов
`xlRowSeparator`	Символ, который используется для разделения строк в массивах символов
`xlAlternateArray-Separator`	Символ, который используется в качестве альтернативного разделителя массивов, если текущий разделитель применен в качестве разделителя десятичной части
`xlDateSeparator`	Разделитель даты (`/`)
`xlTimeSeparator`	Разделитель времени (`:`)
`xlYearCode`	Символ года в формате числа (`y`)
`xlMonthCode`	Символ месяца (`m`)
`xlDayCode`	Символ дня (`d`)
`xlHourCode`	Символ часа (`h`)
`xlMinuteCode`	Символ минуты (`m`)
`xlSecondCode`	Символ секунды (`s`)
`xlCurrencyCode`	Символ валюты
`xlGeneralFormatName`	Название основного формата числа
`xlCurrencyDigits`	Количество десятичных чисел, которые отображаются в формате валюты
`xlCurrencyNegative`	Значение, которое отображает формат валюты для отрицательных значений
`xlNoncurrencyDigits`	Количество десятичных чисел, которые используются в обычных числах
`xlMonthNameChars`	Всегда возвращает три символа для обеспечения обратной совместимости; аббревиатуры месяцев предоставляются Microsoft Windows и могут быть любой длины
`xlWeekdayNameChars`	Всегда возвращает три символа для обеспечения обратной совместимости; аббревиатуры дней недели предоставляются Microsoft Windows и могут быть любой длины
`xlDateOrder`	Целое число, которое отображает порядок элементов в дате
`xl24HourClock`	`True`, если в системе используется 24-часовая система; `False`, если в системе используется 12-часовая система
`xlNonEnglishFunctions`	`True`, если система не отображает функции на английском языке
`xlMetric`	`True`, если используется метрическая система; `False`, если используется британская система измерения
`xlCurrencySpaceBefore`	`True`, если перед символом валюты добавляется пробел
`xlCurrencyBefore`	`True`, если символ валюты добавляется до значения; `False` — в противном случае
`xlCurrencyMinusSign`	`True`, если для представления отрицательного денежного значения используется знак минус; `False` — если для этого применяются скобки
`xlCurrencyTrailingZeros`	`True`, если в нулевых денежных значениях применяются нули, добавленные в конце
`xlCurrencyLeadingZeros`	`True`, если в нулевых денежных значениях применяются нули, добавленные в начале

Окончание табл. 26.3

Константа	Что определяет
xlMonthLeadingZero	True, если при выводе месяца отображается дополняющий нуль (для вывода месяцев применяются числовые значения)
xlDayLeadingZero	True, если при выводе дня недели отображается дополняющий нуль
xl4DigitYears	True, если в системе указываются годы в виде четырех цифр, False — в виде двух цифр
xlMDY	True, если для отображения даты применяется формат месяц-день-год ("длинный" формат даты); False — если дата записана в формате день/месяц/год
xlTimeLeadingZero	True, если при отображении времени используются дополняющие нули

Параметры настройки даты и времени

Если приложение отображает отформатированные даты, которые будут использоваться в других странах, то необходимо удостовериться, что формат даты знаком пользователю. Наиболее удачным подходом считается указание даты с помощью функции VBA DateSerial, что перекладывает бремя форматирования данных "на плечи" Excel (при этом используется краткий формат даты).

Следующая процедура использует функцию DateSerial для присвоения даты переменной StartDate. Дата заносится в ячейку A1 в локальном кратком формате.

```
Sub WriteDate()
   Dim StartDate As Date
   StartDate = DateSerial(2010, 4, 15)
   Range("A1") = StartDate
End Sub
```

Если необходимо дополнительно изменить формат даты, то можно создать код, который будет выполнять эту задачу после записи даты в ячейку. Excel предоставляет несколько именованных форматов даты и времени, а также несколько именованных форматов представления чисел. Все они рассматриваются в справочном руководстве (ключевые фразы для поиска — именованный формат даты/времени или именованный числовой формат).

Управление файлами с помощью VBA

В этой главе...

- Часто выполняемые операции с файлами
- Отображение расширенной информации о файле
- Работа с текстовыми файлами
- Примеры управления текстовыми файлами
- Архивирование и разархивирование файлов
- Модель ADO

В настоящей главе описываются средства VBA, предназначенные для выполнения распространенных операций над любыми файлами, а также для непосредственного управления текстовыми файлами.

Часто выполняемые операции с файлами

Пользователи Excel могут выполнять распространенные операции с файлами двумя способами.

- **Использовать традиционные операторы и функции VBA.** Этот метод применяется во всех версиях Excel.
- **Использовать объект `FileSystemObject`, который задействует библиотеку Microsoft Scripting Library.** Этот метод применяется в версиях Excel 2000 и выше.

Предупреждение

В предыдущих версиях Excel также использовался объект `FileSearch`, который в Excel 2007 и более поздних версиях не поддерживается. Если вызвать на выполнение макрос, использующий объект `FileSearch`, отобразится сообщение об ошибке.

В следующих разделах подробнее рассматриваются упомянутые выше два метода, а также приводится ряд примеров.

Управление файлами с помощью функций VBA

Функции VBA, предназначенные для управления файлами, приведены в табл. 27.1. Большинство из этих функций не требует дополнительных пояснений; все они описаны в справочной системе Excel.

Таблица 27.1. Функции VBA, предназначенные для управления файлами

Функция	Назначение
`ChDir`	Изменяет текущую папку
`ChDrive`	Изменяет текущий диск
`Dir`	Возвращает имя файла или папки, которое соответствует определенному шаблону или атрибуту файла
`FileCopy`	Копирует файл
`FileDateTime`	Возвращает дату и время последнего изменения файла
`FileLen`	Возвращает размер файла (в байтах)
`GetAttr`	Возвращает значение, которое представляет атрибут файла
`Kill`	Удаляет файл
`MkDir`	Создает новую папку
`Name`	Переименовывает файл или папку
`RmDir`	Удаляет пустую папку
`SetAttr`	Изменяет атрибут файла

В оставшейся части раздела рассматриваются примеры, демонстрирующие применение описанных в табл. 27.1 функций.

Определение факта существования файла

Представленная ниже функция возвращает значение `True`, если определенный файл существует. Если файл не существует, функция возвращает значение `False`. Когда функция `Dir` возвращает пустую строку, то файл невозможно найти. В этом случае возвращается значение `False`.

```
Function FileExists(fname) As Boolean
   FileExists = Dir(fname) <> ""
End Function
```

Аргумент функции `FileExists` состоит из полного пути и имени файла. Функция может использоваться на рабочем листе, а также вызываться из кода VBA.

Определение факта существования пути

Следующая функция возвращает значение `True`, если указанный путь существует. В противном случае функция возвращает значение `False`.

```
Function PathExists(pname) As Boolean
'  Возвращает TRUE, если путь существует
   On Error Resume Next
   PathExists = (GetAttr(pname) And vbDirectory) = vbDirectory
End Function
```

Компакт-диск

Функции `FileExists` и `PathExists` находятся на прилагаемом к книге компакт-диске в файле `file functions.xlsm`.

Отображение списка файлов в папке

Следующая процедура отображает (на активном рабочем листе) список файлов, которые содержатся в определенной папке. Кроме того, отображается размер каждого файла, а также дата последнего изменения.

```
Sub ListFiles()
   Dim Directory As String
   Dim r As Long
   Dim f As String
   Directory = "f:\excelfiles\budgeting\"
   r = 1
'  Вставка заголовков
   Cells(r, 1) = "Имя файла"
   Cells(r, 2) = "Размер"
   Cells(r, 3) = "Дата/время"
   Range("A1:C1").Font.Bold = True
'  Получение первого файла
   f = Dir(Directory, vbReadOnly + vbHidden + vbSystem)
   Do While f <> ""
      r = r + 1
      Cells(r, 1) = f
      Cells(r, 2) = FileLen(Directory & f)
      Cells(r, 3) = FileDateTime(Directory & f)
'     Получить следующий файл
      f = Dir()
   Loop
End Sub
```

На рис. 27.1 показан результат выполнения процедуры `ListFiles`.

	A	B	C	D
1	Файлы в D:\Entertaiments\Musik	Размер	Дата/время	
2	01 - 500.mp3	8 581 108	10.09.2009 5:00	
3	02 - Брод.mp3	10 199 660	11.09.2009 18:08	
4	03 - Нога судьбы.mp3	12 020 930	09.09.2009 22:58	
5	04 - Растаманы из глубинки.mp3	9 570 663	11.09.2009 14:30	
6	05 - Брат никотин.mp3	8 214 369	11.09.2009 14:30	
7	06 - Слишком много любви.mp3	8 506 953	10.09.2009 6:31	
8	07 - Псалом 151.mp3	10 633 300	10.09.2009 6:31	
9	08 - Кардиограмма.mp3	8 539 333	10.09.2009 23:02	
10	09 - Северный цвет.mp3	14 886 044	11.09.2009 16:51	
11				

Лист1

Рис. 27.1. Результат выполнения процедуры `ListFiles`

Обратите внимание на то, что процедура использует функцию `Dir` дважды. Первый раз для получения имени первого файла, следующий — для получения остальных имен файлов. Если больше файлов не найдено, функция `Dir` возвращает пустую строку.

Компакт-диск

На прилагаемом к книге компакт-диске в файле `create file list.xlsm` находится версия этой процедуры, позволяющая выбрать папку в диалоговом окне.

Функция `Dir` в качестве первого аргумента принимает символы подстановки. Чтобы получить список (например, файлов Excel), используйте следующий оператор:

```
f = Dir(Directory & "*.xl??", vbReadOnly + vbHidden + vbSystem)
```

Этот оператор выбирает имя первого файла `*.xl??` в указанной папке. Использование символа подстановки приводит к тому, что возвращается четырехсимвольное расширение, которое начинается символами XL. Например, расширение может записываться в виде XLSX, XLTX или XLAM. Второй аргумент функции `Dir` позволяет указывать атрибуты файлов (с применением встроенных констант, например). В рассматриваемом в этом разделе примере функция `Dir` осуществляет выборку имен файлов с отсутствующими атрибутами, файлов "только для чтения", скрытых файлов, а также системных файлов.

В табл. 27.2 приводится перечень встроенных констант для функции `Dir`.

Таблица 27.2. Константы, определяющие атрибуты файлов для функции `Dir`

Константа	Значение	Описание
`vbNormal`	0	Файл без атрибутов. Это настройка, заданная по умолчанию, которая всегда активна
`vbReadOnly`	1	Файлы только для чтения
`vbHidden`	2	Скрытые файлы
`vbSystem`	4	Системные файлы
`vbVolume`	8	Метка тома. Если указан какой-либо другой атрибут, этот атрибут игнорируется
`vbDirectory`	16	Папки. Этот атрибут неработоспособен. Вызов функции `Dir` с атрибутом `vbDirectory` не позволяет постоянно возвращать подпапки

Предупреждение

Если функция `Dir` применяется для циклического обхода файлов, а для управления этими файлами вызывается другая процедура, убедитесь в том, что последняя процедура не использует функцию `Dir`. Активным может быть только единственный "набор" вызовов функции `Dir`.

Применение рекурсивной процедуры для отображения списка файлов во вложенных папках

В этом разделе рассматривается пример функции, которая создает список файлов, находящихся в папке, включающей все подпапки. Необычность этой процедуры заключается в том, что она вызывает саму себя. Подобный процесс называется *рекурсией*.

```
Public Sub RecursiveDir(ByVal CurrDir As String, _
   Optional ByVal Level As Long)
   Dim Dirs() As String
   Dim NumDirs As Long
   Dim FileName As String
   Dim PathAndName As String
```

```
    Dim i As Long
'   Проверка наличия символа обратной черты в конце пути
    If Right(CurrDir, 1) <> "\" Then CurrDir = CurrDir & "\"
'   Помещение заголовков столбцов на активном листе
    Cells(1, 1) = "Путь"
    Cells(1, 2) = "Имя файла"
    Cells(1, 3) = "Размер"
    Cells(1, 4) = "Дата/время"
    Range("A1:D1").Font.Bold = True

'   Получение файлов
    FileName = Dir(CurrDir & "*.*", vbDirectory)
    Do While Len(FileName) <> 0
       If Left(FileName, 1) <> "." Then ' Текущая папка
          PathAndName = CurrDir & FileName
          If (GetAttr(PathAndName) And vbDirectory) = _
             vbDirectory Then
             ' Сохранить найденные папки
             ReDim Preserve Dirs(0 To NumDirs) As String
             Dirs(NumDirs) = PathAndName
             NumDirs = NumDirs + 1
          Else
             ' Запись пути и файла на листе
             Cells(WorksheetFunction.CountA _
                (Range("A:A")) + 1, 1) = CurrDir
             Cells(WorksheetFunction.CountA( _
                Range("B:B")) + 1, 2) = FileName
             Cells(WorksheetFunction.CountA( _
                Range("C:C")) + 1, 3) = FileLen(PathAndName)
             Cells(WorksheetFunction.CountA( _
                Range("D:D")) + 1, 4) = _
             FileDateTime(PathAndName)
       End If
    End If
       FileName = Dir()
    Loop
'   Рекурсивный просмотр найденных папок
    For i = 0 To NumDirs - 1
       RecursiveDir Dirs(i), Level + 2
    Next i
End Sub
```

Процедура использует единственный аргумент `CurrDir`, который представляет собой просматриваемую папку. Информация о каждом файле отображается в активном рабочем листе. В процессе циклического обхода файлов названия подпапок записываются в массив `Dirs`. Если файлы не найдены, процедура вызывает саму себя, причем в качестве аргумента используется элемент массива `Dirs`. После завершения обработки всех папок массива `Dirs` процедура завершает свою работу.

Поскольку процедура `RecursiveDir` использует аргумент, для ее вызова из другой процедуры применяется следующий оператор:

```
Call RecursiveDir("c:\directory\")
```

Компакт-диск

На прилагаемом компакт-диске находится версия этой процедуры, которая позволяет выбирать папку в диалоговом окне. Код этой процедуры находится в файле `recursive file list.xlsm`.

Использование объекта FileSystemObject

Объект FileSystemObject определен в библиотеке Windows Scripting Host и предоставляет доступ к файловой системе компьютера. Этот объект часто применяется в ориентированных на использование сценариев веб-приложениях (например, использующих код VBScript или JavaScript) и поддерживается в Excel 2000 и более поздних версиях программы.

Предупреждение

Зачастую библиотека Windows Scripting Host используется разработчиками вирусов, и в некоторых системах она отключается. Поэтому следует отображать соответствующее предупреждение, если приложение предназначено для использования в разных системах.

Имя FileSystemObject может ввести в заблуждение, поскольку этот объект на самом деле состоит из нескольких объектов, каждый из которых выполняет свои функции.

- **Диск (Drive).** Представляет диск или коллекцию дисков.
- **Файл (File).** Представляет файл или коллекцию файлов.
- **Папка (Folder).** Представляет папку или коллекцию папок.
- **Текстовый поток (TextStream).** Представляет поток текста, который считывается, записывается или добавляется в текстовый файл.

На первом этапе использования объекта FileSystemObject создается его экземпляр. Для этого используется ранее или позднее связывание.

Метод позднего связывания предусматривает использование следующих двух операторов:

```
Dim FileSys As Object
Set FileSys = CreateObject("Scripting.FileSystemObject")
```

Обратите внимание на то, что объявлена объектная переменная FileSys, которая имеет обобщенный тип Object, а не реальный объектный тип. Конкретный тип объекта выполняется в процессе выполнения кода.

Метод раннего связывания, используемый для создания объекта, требует установки ссылки на объектную модель Windows Scripting Host Object Model (рис. 27.2). Для установки подобной ссылки в редакторе VBE выполните команду Tools⇨References (Сервис⇨Ссылки). После установки ссылки создайте объект, воспользовавшись следующими операторами.

```
Dim FileSys As FileSystemObject
Set FileSys = CreateObject("Scripting.FileSystemObject")
```

Благодаря использованию раннего связывания открывается доступ к свойству автоматической вставки объектов (Auto List Members) в VBE. Это свойство облегчает идентификацию применяемых пользователями свойств и методов. Для получения дополнительных сведений относительно объектной модели воспользуйтесь приложением Object Browser (для его вызова нажмите клавишу <F2>).

Рассматриваемые в дальнейшем примеры демонстрируют различные приложения объекта FileSystemObject.

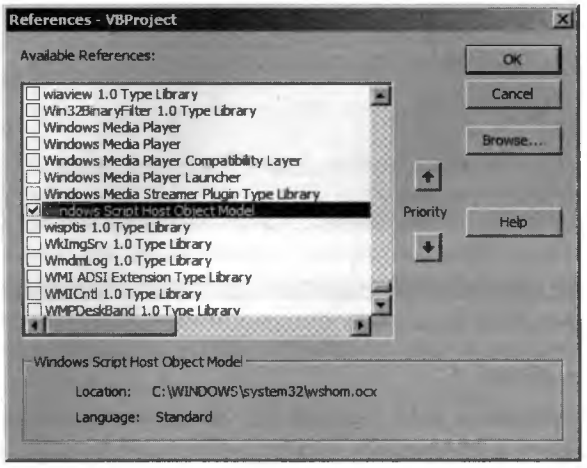


Рис. 27.2. Создание ссылки на объектную модель Windows Script Host Object Model

Определение факта существования файла

Приведенная ниже процедура `Function` принимает один аргумент (путь и имя файла), возвращая значение `True`, если файл существует.

```
Function FileExists3(fname) As Boolean
   Dim FileSys As Object 'FileSystemObject
   Set FileSys = CreateObject("Scripting.FileSystemObject")
   FileExists3 = FileSys.FileExists(fname)
End Function
```

Эта функция создает новый объект `FileSystemObject` под названием `FileSys`, а также получает доступ к свойству `FileExists` этого объекта.

Определение факта существования пути

Приведенная ниже процедура `Function` принимает один аргумент (путь), возвращая значение `True`, если путь существует.

```
Function PathExists2(path) As Boolean
   Dim FileSys As Object 'объект FileSystemObject
   Set FileSys = CreateObject("Scripting.FileSystemObject")
   PathExists2 = FileSys.FolderExists(path)
End Function
```

Отображение сведений о всех доступных дисках в системе

В примере из этого раздела объект `FileSystemObject` применяется для поиска и отображения информации обо всех дисках, установленных в системе. Процедура выполняет циклический обход коллекции `Drives`, а также записывает различные свойства на рабочий лист.

На рис. 27.3 показаны результаты выполнения этой процедуры в системе, состоящей из восьми дисков. Отображается имя диска, состояние его готовности ("ready"), тип диска, имя тома, общий объем, а также размер свободного пространства на диске.

	A	B	C	D	E	F	G
1	Диск	Готов	Тип	Метка тома	Размер	Свободно	
2	A	ЛОЖЬ	Сменный диск				
3	C	ИСТИНА	Жесткий диск		8,002E+10	17439453184	
4	D	ИСТИНА	Жесткий диск		1,6E+11	14097965056	
5	E	ЛОЖЬ	Компакт-диск				
6	F	ЛОЖЬ	Компакт-диск				
7	G	ИСТИНА	Сменный диск		4,004E+09	4004007936	
8	Z	ИСТИНА	Сетевой диск		3,201E+11	2,37136E+11	
9							
10							

Лист1

Рис. 27.3. Результат выполнения процедуры `ShowDriveInfo`

Компакт-диск

Рассматриваемая в этом разделе рабочая книга находится на прилагаемом компакт-диске в файле `show drive info.xlsm`.

```
Sub ShowDriveInfo()
   Dim FileSys As FileSystemObject
   Dim Drv As Drive
   Dim Row As Long
   Set FileSys = CreateObject("Scripting.FileSystemObject")
   Cells.ClearContents
   Row = 1
'  Заголовки столбцов
   Range("A1:F1") = Array("Диск", "Готов", "Тип", "Имя тома", _
      "Размер", "Доступно")
   On Error Resume Next
'  Циклический обход дисков
   For Each Drv In FileSys.Drives
      Row = Row + 1
      Cells(Row, 1) = Drv.DriveLetter
      Cells(Row, 2) = Drv.IsReady
      Select Case Drv.DriveType
         Case 0: Cells(Row, 3) = "Неизвестно"
         Case 1: Cells(Row, 3) = "Сменный диск"
         Case 2: Cells(Row, 3) = "Жесткий диск"
         Case 3: Cells(Row, 3) = "Сетевой диск"
         Case 4: Cells(Row, 3) = "Компакт-диск"
         Case 5: Cells(Row, 3) = "RAM-диск"
      End Select
      Cells(Row, 4) = Drv.VolumeName
      Cells(Row, 5) = Drv.TotalSize
      Cells(Row, 6) = Drv.AvailableSpace
   Next Drv
'  Создание таблицы
   ActiveSheet.ListObjects.Add xlSrcRange, _
      Range("A1").CurrentRegion, , xlYes
End Sub
```

Перекрестная ссылка

В главе 11 описан еще один метод получения информации о дисках с помощью функций Windows API.

Отображение расширенной информации о файле

В примере из этого раздела отображается расширенный набор свойств файла, находящегося в определенной папке. Отображаемая информация зависит от выбранного типа файла. Например, файлы изображений обладают такими свойствами, как `Camera Model` (Модель камеры) и `Dimensions` (Размеры); звуковые файлы имеют такие свойства, как `Artist` (Исполнитель), `Title` (Название), `Duration` (Длительность) и т.д.

Фактический набор свойств зависит от используемой версии Windows. Версия Windows Vista поддерживает 267 свойств, а Windows 7 — еще больше. Ниже представлен код процедуры, создающий список свойств файла для активного рабочего листа.

```
Sub ListFileProperties()
   Dim i As Long
   Dim objShell As Object 'IShellDispatch4
   Dim objFolder As Object 'Folder3
'  Создание объекта
   Set objShell = CreateObject("Shell.Application")
'  Указание любой папки
   Set objFolder = objShell.Namespace("C:\")
'  Перечень свойств
   For i = 0 To 500
      Cells(i + 1, 1) = _
         objFolder.GetDetailsOf(objFolder.Items, i)
   Next i
End Sub
```

Предупреждение

К сожалению, значения свойств отличаются в различных версиях Windows. Например, в Windows 2000 свойство `Title` имеет значение 11. В Windows XP этому свойству присуще значение 10, а в Windows Vista — значение 21.

Ниже представлена процедура `FileInfo`, работа которой основана на объекте Windows `Shell.Application`. В процессе выполнения этой процедуры запрашивается название папки (с помощью функции `GetDirectory`, код которой здесь не показан), а затем выводятся сведения, описывающие 41 свойство (и это далеко не все) для каждого файла в папке.

```
Sub FileInfo()
   Dim c As Long, r As Long, i As Long
   Dim FileName As Object 'FolderItem2
   Dim objShell As Object 'IShellDispatch4
   Dim objFolder As Object 'Folder3

'  Создание объекта
   Set objShell = CreateObject("Shell.Application")

'  Запрос названия папки
   Set objFolder = objShell.Namespace(GetDirectory)
'  Вставка заголовков на активном листе
   Worksheets.Add
   c = 0
   For i = 0 To 40
      c = c + 1
```

```
      Cells(1, c) = objFolder.GetDetailsOf(objFolder.Items, i)
   Next i
'  Циклический обход файлов
   r = 1
   For Each FileName In objFolder.Items
      c = 0
      r = r + 1
      For i = 0 To 40
         c = c + 1
         Cells(r, c) = objFolder.GetDetailsOf(FileName, i)
      Next i
   Next FileName
'  Создание таблицы
   ActiveSheet.ListObjects.Add xlSrcRange, _
      Range("A1").CurrentRegion
End Sub
```

На рис. 27.4 показана часть информации, выводимой этой процедурой, в случае выбора папки со звуковыми файлами в формате MP3.

	A	B	C	D	E	F	G	H	I	J	K	L
1	Имя	Размер	Тип	Изменен	Дата создания	Открыт	Атрибуты	Состояние	Владелец	Автор	Заголовок	Тема
2	01 - Дорога в рай	6 135 КБ	Звук в формате MP3	15.07.2000 13:50	13.12.2009 3:23	26.04.2010	A	Подключен	SERG-7C4485C3C7\Alex			
3	02 - Скорее пой	3 318 КБ	Звук в формате MP3	15.07.2000 13:51	13.12.2009 3:23	26.04.2010	A	Подключен	SERG-7C4485C3C7\Alex			
4	03 - Магия чисел	3 200 КБ	Звук в формате MP3	15.07.2000 13:52	13.12.2009 3:23	26.04.2010	A	Подключен	SERG-7C4485C3C7\Alex			
5	04 - На хрена нам война	2 925 КБ	Звук в формате MP3	15.07.2000 13:52	13.12.2009 3:23	26.04.2010	A	Подключен	SERG-7C4485C3C7\Alex			
6	05 - Поцелуй в глаза	2 240 КБ	Звук в формате MP3	15.07.2000 13:53	13.12.2009 3:23	26.04.2010	A	Подключен	SERG-7C4485C3C7\Alex			
7	06 - Все мы пойдём вслед за дяка	4 296 КБ	Звук в формате MP3	15.07.2000 13:54	13.12.2009 3:23	26.04.2010	A	Подключен	SERG-7C4485C3C7\Alex			
8	07 - Женщина	4 070 КБ	Звук в формате MP3	15.07.2000 13:55	13.12.2009 3:23	26.04.2010	A	Подключен	SERG-7C4485C3C7\Alex			
9	08 - Раста это классно	4 309 КБ	Звук в формате MP3	15.07.2000 13:56	13.12.2009 3:23	26.04.2010	A	Подключен	SERG-7C4485C3C7\Alex			
10	09 - Мариш-на-фарш	2 666 КБ	Звук в формате MP3	15.07.2000 13:57	13.12.2009 3:23	26.04.2010	A	Подключен	SERG-7C4485C3C7\Alex			
11	10 - Дяка пустит трамвай из болота в рай	6 104 КБ	Звук в формате MP3	15.07.2000 13:59	13.12.2009 3:23	26.04.2010	A	Подключен	SERG-7C4485C3C7\Alex			
12	11 - Если Дяка хочет, то Дяка хлопочет	3 966 КБ	Звук в формате MP3	15.07.2000 14:00	13.12.2009 3:23	26.04.2010	A	Подключен	SERG-7C4485C3C7\Alex			
13	Копия 01 - Дорога в рай	6 135 КБ	Звук в формате MP3	15.07.2000 13:50	13.12.2009 3:23	26.04.2010	A	Подключен	SERG-7C4485C3C7\Alex			
14	Копия 04 - На хрена нам война	2 925 КБ	Звук в формате MP3	15.07.2000 13:52	13.12.2009 3:23	26.04.2010	A	Подключен	SERG-7C4485C3C7\Alex			

Рис. 27.4. Сведения о файлах, находящихся в выбранной папке

В этом примере используется позднее связывание для создания объекта `Shell.Application`, в результате чего объекты объявляются обобщенным образом. Для применения раннего связывания используйте команду VBE Tools⇨References (Сервис⇨Ссылки) и создайте ссылку на объект Microsoft Shell Controls and Automation.

Компакт-диск

Рассматриваемый в этом разделе пример находится на прилагаемом компакт-диске в файле `file information.xlsm`.

Работа с текстовыми файлами

VBA содержит ряд операторов, которые позволяют управлять файлами на низком уровне. Эти операторы ввода-вывода предоставляют разработчику более широкие возможности, чем стандартные средства экспорта и импорта файлов Excel.

Доступ к файлу можно осуществлять в одном из трех режимов.

- **Последовательный доступ.** Этот режим является самым распространенным. Он позволяет считывать и записывать отдельные символы или целые строки данных.
- **Произвольный доступ.** Режим, используемый только при программировании приложений баз данных (не рекомендуется использовать в VBA, поскольку для этого существуют более подходящие средства).

- **Двоичный доступ.** Используется для чтения и записи с любой позиции в файле с точностью до байта. Данный режим может применяться для сохранения или отображения растрового изображения. В VBA он практически не используется.

По причине того что произвольный и двоичный режимы доступа в VBA используются редко, в данной главе основное внимание будет уделено рассмотрению последовательного доступа. При последовательном доступе файл считывается последовательно. Другими словами, приложение начинает чтение с начала файла и последовательно получает строку за строкой. При записи файла приложение добавляет строки в конец существующего файла.

Примечание

Для чтения и записи данных в этой главе используется традиционный метод "канала данных". Также можно воспользоваться объектным подходом. Объект `FileSystemObject` включает объект `TextStream`, который может применяться для чтения и записи текстовых файлов. Объект `FileSystemObject` является составной частью библиотеки Windows Scripting Host. Как упоминалось ранее, эта служба поддержки сценариев в большинстве систем отключена по причине ее склонности к распространению вирусов.

Открытие текстового файла

Функция VBA `Open` (не путайте с методом `Open` объекта `Application`) используется при открытии файла для чтения или записи. Перед тем как начать чтение или запись файла, его необходимо открыть.

Функция `Open` является очень гибким, поэтому она имеет относительно сложный синтаксис.

```
Open путь For режим [Access доступ] [ блокировка] _
   As [#] номер_файла [Len= длина_записи]
```

- *Путь* (обязательный параметр). Аргумент *путь* оператора `Open` чрезвычайно прост. Он включает имя и путь (необязательно) открываемого файла.
- *Режим* (обязательный параметр). При открытии файлов используется один из следующих режимов:
  - `Append` — режим последовательного доступа, который позволяет осуществлять чтение файла, а также добавлять данные в конец файла;
  - `Input` — режим последовательного доступа, который позволяет читать файл, но не дает возможности записывать в него данные;
  - `Output` — режим последовательного доступа, который позволяет выполнять чтение и запись файла. В этом режиме всегда создается новый файл (существующий файл с текущим именем удаляется);
  - `Binary` — режим произвольного доступа, в котором производится побайтовое чтение и запись данных;
  - `Random` — режим произвольного доступа, который позволяет выполнять чтение и запись данных блоками, установленными значением аргумента *длина_записи* оператора `Open`.

- *Доступ*. Этот аргумент определяет допустимые операции над файлом (необязательный параметр). Этот параметр может иметь значения `Read`, `Write` и `Read Write`.
- *Блокировка*. Этот аргумент используется для разрешения проблем многопользовательского доступа (необязательный параметр). Этот параметр может иметь следующие значения: `Shared`, `Lock Read`, `Lock Write`, а также `Lock Read Write`.
- *Номер_файла*. Значение этого обязательного параметра находится в пределах от `1` до `511`. Для получения следующего свободного номера файла можно использовать функцию `FreeFile`. (Подробное описание этой функции можно найти в разделе "Получение номера файла".)
- *Длина_записи*. Этот параметр определяет длину записи (для файлов с произвольным доступом) или размер буфера (для файлов с последовательным доступом) и является необязательным.

Чтение текстового файла

Базовой процедурой чтения текстового файла с использованием VBA является следующая последовательность действий.

1. Открытие файла с помощью функции `Open`.
2. Указание позиции в файле с помощью функции `Seek` (не обязательно).
3. Чтение данных из файла с помощью операторов `Input`, `Input #` или `Line Input #`.
4. Закрытие файла с помощью функции `Close`.

Запись в текстовый файл

Базовая процедура записи текстового файла состоит из следующей последовательности действий.

1. Открытие или создание файла с помощью функции `Open`.
2. Указание позиции в файле с помощью функции `Seek` (не обязательно).
3. Запись данных в файл с помощью операторов `Write #` или `Print #`.
4. Закрытие файла с помощью функции `Close`.

Получение номера файла

Большинство программистов, использующих VBA, просто назначают номер файла при вызове функции `Open`.

```
Open "myfile.txt" For Input As #1
```

Теперь в следующих операторах на этот файл можно ссылаться, как на `#1`.

Если открыт первый файл и вы открываете второй, то последний получит номер `#2`.

```
Open "another.txt" For Input As #2
```

Еще одним методом является использование функции VBA `FreeFile`, которая позволяет получить свободный дескриптор файла. После этого на файл можно ссылаться с помощью переменной. Приведем пример использования этой функции.

```
FileHandle = FreeFile
Open "myfile.txt" For Input As FileHandle
```

Определение или установка позиции в файле

Для последовательного доступа к файлу редко возникает необходимость в получении текущей позиции. Но если по определенной причине такая информация нужна, то можно воспользоваться функцией `Seek`.

Операторы чтения и записи в файл

В VBA имеются несколько операторов чтения и записи данных в файл. Представленные далее операторы используются для чтения данных из файла с последовательным доступом:

- `Input` — читает из файла указанное количество символов;
- `Input #` — читает файл в виде последовательности переменных; переменные разделяются запятой;
- `Line Input #` — читает файл построчно (строки разделяются символами возврата каретки и/или перевода строки).

Для записи данных в файл с последовательным доступом используются два оператора.

- `Write #` — записывает последовательность значений, в которой каждое значение отделено запятой и находится в кавычках. Если оператор завершается точкой с запятой, значения не будут разделяться символами "возврат каретки/перевод строки". Данные, записанные с помощью оператора `Write #`, обычно читаются из файла с помощью оператора `Input #`.
- `Print #` — записывает последовательность значений, в которой каждое значение отделено символом пробела. Если завершить оператор точкой с запятой, то после каждого значения не будет вставляться последовательность "возврат каретки/перевод строки". Данные, записанные с помощью оператора `Print #`, обычно читаются с помощью оператора `Line Input #` или `Input #`.

Средства Excel импорта и экспорта файлов

В Excel поддерживаются три типа текстовых файлов.

- **Файлы CSV (Comma Separated Values — разделенные запятыми значения).** Столбцы данных разделяются запятыми, а каждая строка данных завершается символом возврата каретки. В некоторых неанглийских версиях Excel вместо запятой используется точка с запятой.
- **Файлы PRN.** Столбцы данных выравниваются по определенному символу, а каждая строка завершается символом возврата каретки. Эти файлы также называются *файлами постоянной ширины*.
- **Файлы TXT (Tab delimited — разделенные символами табуляции).** Столбцы данных разделены с помощью символов табуляции, а каждая строка данных завершается символом возврата каретки.

При попытке открытия текстового файла с помощью команды Файл⇨Открыть (File⇨Open) открывается мастер импорта текста, с помощью которого можно определить способ разделения столбцов. Если в тексте в качестве разделителя используется сим-

вол табуляции или запятая, Excel открывает файл без отображения мастера импорта текста. Если данные интерпретируются некорректно, закройте файл и попробуйте переименовать его, указав расширение .TXT.

Мастер распределения текста по столбцам (вызывается с помощью команды Данные⇨Работа с данными⇨Текст по столбцам (Data⇨Data Tools⇨Text to Table)) напоминает мастер импорта текста. Отличие заключается в том, что он работает с данными, находящимися в единственном столбце рабочего листа.

Примеры управления текстовыми файлами

В данном разделе содержится несколько примеров, демонстрирующих различные способы управления текстовыми файлами.

Импортирование данных из текстового файла

В следующем примере процедура читает текстовый файл и размещает каждую строку данных в отдельную ячейку (начиная с активной ячейки).

```
Sub ImportData()
   Open "c:\data\textfile.txt" For Input As #1
   r = 0
   Do Until EOF(1)
      Line Input #1, data
      ActiveCell.Offset(r, 0) = data
      r = r + 1
   Loop
   Close #1
End Sub
```

В большинстве случаев данная процедура не принесет пользы, так как каждая строка данных просто помещается в отдельную ячейку. Часто бывает проще открыть файл, воспользовавшись командой Файл⇨Открыть (File⇨Open).

Экспортирование диапазона в текстовый файл

Рассматриваемая в этом разделе процедура записывает данные из выбранного диапазона рабочего листа в текстовый файл CSV. Как известно, Excel может экспортировать данные в CSV-файл, но при этом экспортируется весь рабочий лист. Приведенный ниже макрос работает с указанным диапазоном ячеек.

```
Sub ExportRange()
   Dim Filename As String
   Dim NumRows As Long, NumCols As Integer
   Dim r As Long, c As Integer
   Dim Data
   Dim ExpRng As Range
   Set ExpRng = Selection
   NumCols = ExpRng.Columns.Count
   NumRows = ExpRng.Rows.Count
   Filename = Application.DefaultFilePath & "\textfile.csv"
   Open Filename For Output As #1
   For r = 1 To NumRows
      For c = 1 To NumCols
         Data = ExpRng.Cells(r, c).Value
         If IsNumeric(Data) Then Data = Val(Data)
         If IsEmpty(ExpRng.Cells(r, c)) Then Data = ""
```

```
        If c <> NumCols Then
           Write #1, Data;
        Else
           Write #1, Data
        End If
     Next c
  Next r
  Close #1

End Sub
```

Обратите внимание на то, что процедура использует два оператора `Write #`. Первый завершается точкой с запятой, поэтому последовательность "возврат каретки/перевод строки" в файл не добавляется. Но для последней ячейки в строке второй оператор `Write #` не имеет точки с запятой, что приводит к завершению строки и добавлению следующей ячейки уже в новой строке.

Переменная `Data` используется для хранения содержимого каждой ячейки. Если ячейка имеет числовой формат, то переменная принимает значение, что обеспечивает отсутствие кавычек при сохранении данных. Если ячейка пуста, то ее свойство `Value` возвращает значение `0`. Таким образом, проверяются пустые ячейки (для этого используется функция `IsEmpty`), и вместо `0` подставляется пустая строка.

На рис. 27.5 показано содержимое результирующего файла, просматриваемого в окне текстового редактора Windows Notepad.

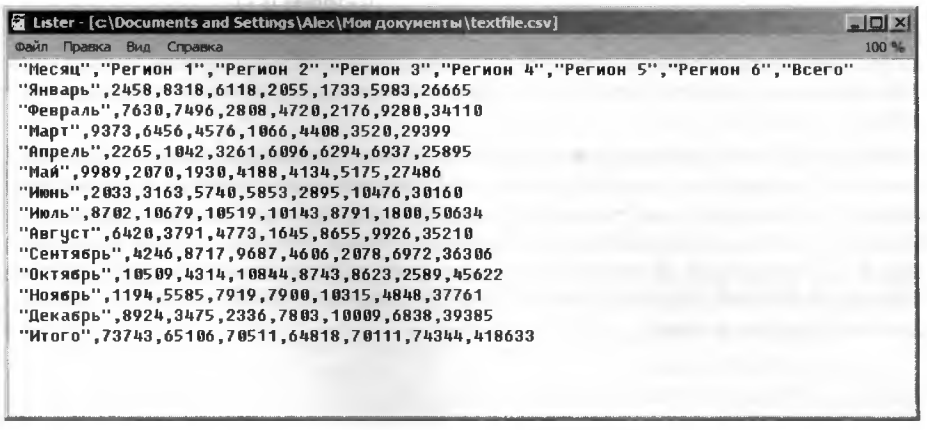


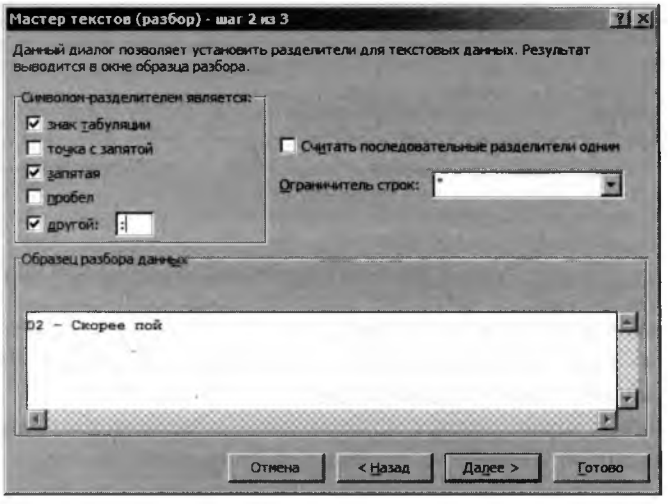
Рис. 27.5. Этот текстовый файл создан с помощью VBA

Компакт-диск

Примеры из этого и следующего разделов находятся на прилагаемом к книге компакт-диске в файле `export and import csv.xlsm`.

Почему Excel некорректно анализирует данные

Независимо от того, имеете ли вы дело с импортированным CSV-файлом либо данные вставляются непосредственно в рабочий лист, Excel может выполнять разбиение данных некорректно. И причина этого заключается в некорректной работе мастера распределения текста по столбцам. Обратите внимание на следующий рисунок, иллюстрирующий второй этап работы мастера, распределяющего единственный столбец, который содержит разграниченные данные, по многим столбцам.



В рассматриваемом случае применяются три различных разделителя: символ табуляции, запятая и двоеточие.

Разбиение текста на отдельные столбцы полезно во многих случаях. Проблема появляется в том случае, когда Excel пытается запомнить настройки в целях их применения для выполнения в дальнейшем операций по импорту и вставке файлов в формате CSV. Иногда эта методика оказывается полезной, но чаще всего от нее мало толку. Для очистки ранее сохраненных символов-разделителей отобразите показанное на рисунке диалоговое окно, отмените выбранные ранее символы-разделители, а затем щелкните на кнопке Отмена (Cancel).

Если импорт либо вставка данных производится с помощью макроса, в этом случае невозможно непосредственно проверить установленные символы-разделители либо вернуться к стандартным разделителям. В этом случае можно воспользоваться "имитацией" операции разбиения текста по столбцам. Именно этим и занимается указанная ниже процедура. В частности, она создает эффект отмены всех настроек в диалоговом окне Мастер текстов (разбор) (Text To Columns), не внося при этом каких-либо изменений в рабочую книгу.

```
Sub ClearTextToColumns()
   On Error Resume Next
   If IsEmpty(Range("A1")) Then Range("A1") = "XYZZY"
   Range("A1").TextToColumns Destination:=Range("A1"), _
      DataType:=xlDelimited, _
      TextQualifier:=xlDoubleQuote, _
      ConsecutiveDelimiter:=False, _
      Tab:=False, _
      Semicolon:=False, _
      Comma:=False, _
      Space:=False, _
      Other:=False, _
      OtherChar:=""
   If Range("A1") = "XYZZY" Then Range("A1") = ""
   If Err.Number <> 0 Then MsgBox Err.Description
End Sub
```

При выполнении макроса предполагается, что рабочий лист активен и не защищен от изменений. Обратите внимание на то, что содержимое ячейки A1 не может изме-

няться, поскольку для метода TextToColumns не указаны какие-либо операции. Если ячейка A1 пустая, код вставляет временную строку (поскольку метод TextToColumns неработоспособен, если ячейка пустая). Перед завершением работы процедура удаляет временную строку.

Импортирование текстового файла в диапазон

Рассматриваемый в этом разделе пример читает содержимое текстового файла, который был создан в предыдущем примере. После этого прочитанные значения сохраняются начиная с активной ячейки. Приложение читает каждый символ и разбирает полученные строки данных. При этом символы кавычек игнорируются, а запятые используются для разделения столбцов.

```
Sub ImportRange()
   Dim ImpRng As Range
   Dim Filename As String
   Dim r As Long, c As Integer
   Dim txt As String, Char As String * 1
   Dim Data
   Dim i As Integer

   Set ImpRng = ActiveCell
   On Error Resume Next
   Filename = Application.DefaultFilePath & "\textfile.csv"
   Open Filename For Input As #1
   If Err <> 0 Then
      MsgBox "Не найден: " & Filename, vbCritical, "Ошибка"
      Exit Sub
   End If
   r = 0
   c = 0
   txt = ""
   Application.ScreenUpdating = False
   Do Until EOF(1)
      Line Input #1, Data
      For i = 1 To Len(Data)
         Char = Mid(Data, i, 1)
         If Char = "," Then ' запятая
            ActiveCell.Offset(r, c) = txt
            c = c + 1
            txt = ""
         ElseIf i = Len(Data) Then ' конец строки
            If Char <> Chr(34) Then txt = txt & Char
            ActiveCell.Offset(r, c) = txt
            txt = ""
         ElseIf Char <> Chr(34) Then
            txt = txt & Char
         End If
      Next i
      c = 0
      r = r + 1
   Loop
   Close #1
   Application.ScreenUpdating = True
End Sub
```

Примечание

Описанная выше процедура имеет один недостаток: она не может импортировать такие значения, как символы кавычек или запятой. Кроме того, импортированные значения даты окружены символами решетки, например `#2007-05-12#`.

Протоколирование операций в Excel

Пример, приведенный в этом разделе, предназначен для записи данных в текстовый файл при каждом открытии и закрытии Excel. Для того чтобы эта процедура работала надежно, она должна находиться в рабочей книге, которая всегда открывается и закрывается вместе с Excel. Рекомендуется сохранять эту процедуру в персональной книге макросов.

Следующая процедура, которая находится в модуле кода объекта `ThisWorkbook` (ЭтаКнига), выполняется каждый раз при открытии файла.

```
Private Sub Workbook_Open()
   Open Application.Path & "\excelusage.txt" For Append As #1
   Print #1, "Запущено " & Now
   Close #1
End Sub
```

В результате выполнения этой процедуры в файл `excelusage.txt` добавляется новая строка. Эта строка содержит текущие значения даты и времени и может выглядеть следующим образом:

```
Запущено 11/16/10 9:27:43 PM
```

Следующая процедура вызывается перед закрытием рабочей книги. В результате ее выполнения добавляется слово `Остановлено`, а также значения даты и времени.

```
Private Sub Workbook_BeforeClose(Cancel As Boolean)
   Open Application.Path & "\excelusage.txt" _
      For Append As #1
   Print #1, "Остановлено " & Now
   Close #1
End Sub
```

Компакт-диск

Рабочая книга под названием `excel usage log.xslm` включает все описанные в разделе процедуры и находится на прилагаемом компакт-диске.

Перекрестная ссылка

Обратитесь к главе 19, где содержатся дополнительные сведения о процедурах обработки событий, таких как `Workbook_Open` и `Workbook_BeforeClose`.

Фильтрация текстового файла

Пример данного раздела демонстрирует одновременную работу с двумя текстовыми файлами. Процедура `FilterFile`, которая приводится ниже, читает текстовый файл (`infile.txt`) и копирует строки, содержащие определенный текст, во второй текстовый файл (`output.txt`).

```
Sub FilterFile()
   Open ThisWorkbook.Path & "\infile.txt" For Input As #1
```

```
   Open Application.DefaultFilePath & "\output.txt" _
      For Output As #2
   TextToFind = "Январь"
   Do Until EOF(1)
      Line Input #1, data
      If InStr(1, data, TextToFind) Then
         Print #2, data
      End If
   Loop
   Close #1
End Sub
```

Компакт-диск

Пример, рассматриваемый в этом разделе, находится на прилагаемом к книге компакт-диске в файле `filter text file.xlsm`.

Экспортирование диапазона в формат HTML

Рассматриваемый в этой главе пример демонстрирует порядок экспортирования диапазона в файл HTML. Этот файл фактически представляет собой текстовый файл, включающий специальные теги форматирования, описывающие способ представления информации в окне веб-браузера.

Что мешает использовать команду Excel Файл⇨Сохранить как веб-страницу (File⇨Save As)? Процедура, приведенная в данном примере, имеет одно заметное преимущество: она не создает "раздутый" код HTML. Например, процедура `ExportToHTML` использовалась для экспорта диапазона из 70 ячеек. Результирующий файл имеет размер 2,6 Кбайт. Если выполнить команду Excel Файл⇨Сохранить как веб-страницу (File⇨Save as Web Page), то результирующий файл будет иметь размер 15,8 Кбайт (в шесть раз больше).

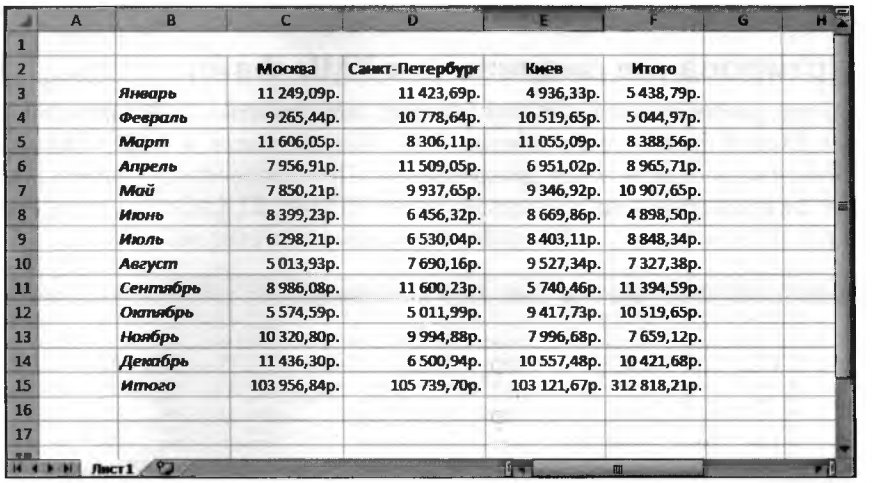
С другой стороны, процедура `ExportToHTML` не сохраняет все параметры форматирования ячеек. Она поддерживает только форматирование с помощью полужирного и курсивного начертания, а также выравнивание по горизонтали. Эта процедура также может применяться во многих ситуациях, а также служить основой для улучшений в дальнейшем.

```
Sub ExportToHTML()
   Dim Filename As Variant
   Dim TDOpenTag As String, TDCloseTag As String
   Dim CellContents As String
   Dim Rng As Range
   Dim r As Long, c As Integer

'  Использование выделенного диапазона ячеек
   Set Rng = Application.Intersect(ActiveSheet.UsedRange, _
      Selection)
   If Rng Is Nothing Then
      MsgBox "Нечего экспортировать.", vbCritical
      Exit Sub
   End If
'  Получить имя файла
   Filename = Application.GetSaveAsFilename( _
      InitialFileName:="myrange.htm", _
      fileFilter:="HTML Files(*.htm), *.htm")
      If Filename = False Then Exit Sub
```

```
'  Открытие текстового файла
   Open Filename For Output As #1

'  Запись тегов
   Print #1, "<HTML>"
   Print #1, "<TABLE BORDER=0 CELLPADDING=3>"

'  Циклический обход ячеек
   For r = 1 To Rng.Rows.Count
      Print #1, "<TR>"
      For c = 1 To Rng.Columns.Count
         Select Case Rng.Cells(r, c).HorizontalAlignment
         Case xlHAlignLeft
            TDOpenTag = "<TD ALIGN=LEFT>"
         Case xlHAlignCenter
            TDOpenTag = "<TD ALIGN=CENTER>"
         Case xlHAlignGeneral
            If IsNumeric(Rng.Cells(r, c)) Then
               TDOpenTag = "<TD ALIGN=RIGHT>"
            Else
               TDOpenTag = "<TD ALIGN=LEFT>"
            End If
         Case xlHAlignRight
            TDOpenTag = "<TD ALIGN=RIGHT>"
         End Select

         TDCloseTag = "</TD>"
         If Rng.Cells(r, c).Font.Bold Then
            TDOpenTag = TDOpenTag & "<B>"
            TDCloseTag = "</B>" & TDCloseTag
         End If
         If Rng.Cells(r, c).Font.Italic Then
            TDOpenTag = TDOpenTag & "<I>"
            TDCloseTag = "</I>" & TDCloseTag
         End If

         CellContents = Rng.Cells(r, c).Text
         Print #1, TDOpenTag & CellContents & TDCloseTag
      Next c
      Print #1, "</TR>"
   Next r
'  Закрытие таблицы
   Print #1, "</TABLE>"
   Print #1, "</HTML>"
'  Закрытие файла
   Close #1

'  Сообщение для пользователя
   MsgBox Rng.Count & " Экспортируемые ячейки " & Filename
End Sub
```

Процедура начинает свою работу с определения экспортируемого диапазона. Данный диапазон определяется как пересечение выделенного диапазона и используемой области рабочего листа. Это позволяет удостовериться, что не будет обрабатываться весь столбец или строка. Пользователь видит на экране диалоговое окно с просьбой указать имя файла. Далее открывается указанный текстовый файл. Основная работа выполняется в двух циклах `For-Next`. Код генерирует соответствующий код HTML (записывает необходимые дескрипторы) и заносит в файл экспортируемые данные. Наконец, файл закрывается, и пользователь может увидеть окно сообщения с итоговой информацией.

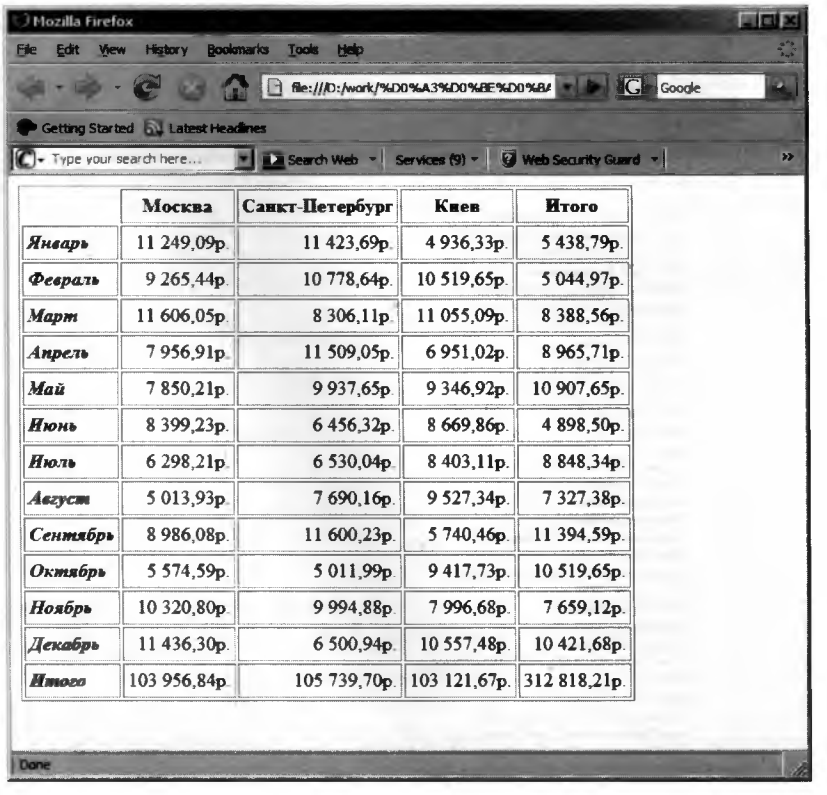
На рис. 27.6 показан диапазон данных рабочего листа, а на рис. 27.7 можно посмотреть, как он выглядит в окне веб-браузера после преобразования в HTML-код.



	A	B	C	D	E	F	G	H
1								
2			Москва	Санкт-Петербург	Киев	Итого		
3		Январь	11 249,09р.	11 423,69р.	4 936,33р.	5 438,79р.		
4		Февраль	9 265,44р.	10 778,64р.	10 519,65р.	5 044,97р.		
5		Март	11 606,05р.	8 306,11р.	11 055,09р.	8 388,56р.		
6		Апрель	7 956,91р.	11 509,05р.	6 951,02р.	8 965,71р.		
7		Май	7 850,21р.	9 937,65р.	9 346,92р.	10 907,65р.		
8		Июнь	8 399,23р.	6 456,32р.	8 669,86р.	4 898,50р.		
9		Июль	6 298,21р.	6 530,04р.	8 403,11р.	8 848,34р.		
10		Август	5 013,93р.	7 690,16р.	9 527,34р.	7 327,38р.		
11		Сентябрь	8 986,08р.	11 600,23р.	5 740,46р.	11 394,59р.		
12		Октябрь	5 574,59р.	5 011,99р.	9 417,73р.	10 519,65р.		
13		Ноябрь	10 320,80р.	9 994,88р.	7 996,68р.	7 659,12р.		
14		Декабрь	11 436,30р.	6 500,94р.	10 557,48р.	10 421,68р.		
15		Итого	103 956,84р.	105 739,70р.	103 121,67р.	312 818,21р.		
16								
17								

Лист1

Рис. 27.6. Диапазон данных рабочего листа, подготовленный для преобразования в формат HTML



Mozilla Firefox

File Edit View History Bookmarks Tools Help

file:///D:/work/%D0%A3%D0%BE%D0%B

Google

Getting Started Latest Headlines

Type your search here... Search Web Services (9) Web Security Guard

	Москва	Санкт-Петербург	Киев	Итого
Январь	11 249,09р.	11 423,69р.	4 936,33р.	5 438,79р.
Февраль	9 265,44р.	10 778,64р.	10 519,65р.	5 044,97р.
Март	11 606,05р.	8 306,11р.	11 055,09р.	8 388,56р.
Апрель	7 956,91р.	11 509,05р.	6 951,02р.	8 965,71р.
Май	7 850,21р.	9 937,65р.	9 346,92р.	10 907,65р.
Июнь	8 399,23р.	6 456,32р.	8 669,86р.	4 898,50р.
Июль	6 298,21р.	6 530,04р.	8 403,11р.	8 848,34р.
Август	5 013,93р.	7 690,16р.	9 527,34р.	7 327,38р.
Сентябрь	8 986,08р.	11 600,23р.	5 740,46р.	11 394,59р.
Октябрь	5 574,59р.	5 011,99р.	9 417,73р.	10 519,65р.
Ноябрь	10 320,80р.	9 994,88р.	7 996,68р.	7 659,12р.
Декабрь	11 436,30р.	6 500,94р.	10 557,48р.	10 421,68р.
Итого	103 956,84р.	105 739,70р.	103 121,67р.	312 818,21р.

Done

Рис. 27.7. Так выглядят данные рабочего листа после преобразования в формат HTML

Компакт-диск

Рассматриваемый в этом разделе пример находится на прилагаемом к книге компакт-диске в файле `export to HTML.xlsm`.

Экспортирование диапазона в XML-файл

В примере, рассматриваемом в этом разделе, выполняется экспортирование диапазона данных Excel в простой файл данных XML. Как вы знаете, в XML каждый элемент данных помечается собственным дескриптором. Процедура этого примера использует в качестве дескрипторов подписи первой строки. На рис. 27.8 показан исходный диапазон, а на рис. 27.9 — готовый XML-файл, отображенный в окне веб-браузера.

EmployeeID	LastName	FirstName	Title	BirthDate	HireDate	Address	City	Region	PostalCode	Country	HomePhone
9001	Davolio	Nancy	Sales Representative	12.8.68	5.1.01	507 - 20th Ave. E. Apt. 2A	Seattle	WA	98122	USA	(206) 555-9857
9002	Fuller	Andrew	Vice President, Sales	2.19.52	8.14.02	908 W. Capital Way	Tacoma	WA	98401	USA	(206) 555-9482
9003	Leverling	Janet	Sales Representative	8.30.63	4.1.06	722 Moss Bay Blvd.	Kirkland	WA	98033	USA	(206) 555-3412
9004	Peacock	Margaret	Sales Representative	9.19.58	5.3.99	4110 Old Redmond Rd.	Redmond	WA	98052	USA	(206) 555-8122
9005	Buchanan	Steven	Sales Manager	3.4.55	10.17.93	14 Garrett Hill	London		SW1 8JR	UK	(71) 555-4848
9006	Suyama	Michael	Sales Representative	7.2.63	10.17.97	Coventry House Miner Rd.	London		EC2 7JR	UK	(71) 555-7773
9007	King	Robert	Sales Representative	5.29.60	1.2.98	Edgeham Hollow Winchester Way	London		RG1 9SP	UK	(71) 555-5598
9008	Callahan	Laura	Inside Sales Coordinator	1.9.58	3.5.94	4726 - 11th Ave. N.E.	Seattle	Wa	98105	USA	(206) 555-1189
9009	Dodsworth	Ann	Sales Representative	7.2.69	11.15.03	7 Houndstooth Rd.	London		WG2 7LT	UK	(71) 555-4444
9102	Jackson	Raymond	Sales Representative	2.16.52	3.4.00	11 Franklin Way	Portland	OR	97223	USA	(503)555-0983

Рис. 27.8. Данные из этого диапазона будут преобразованы в формат XML

Рис. 27.9. Так выглядят данные рабочего листа после преобразования в формат XML

Примечание

Хотя, начиная с версии Excel 2003, появилась улучшенная поддержка XML-файлов, даже в Excel 2010 невозможно создание XML-файла на основе произвольного диапазона данных без наличия файла карты (схемы) для данных.

Ниже приведен код процедуры ExportToXML. Как видите, она имеет много общего с процедурой ExportToHTML, рассмотренной в предыдущем разделе.

```
Sub ExportToXML()
   Dim Filename As Variant
   Dim Rng As Range
   Dim r As Long, c As Long

'  Настройка диапазона
   Set Rng = Range("Table1[#All]")
'  Получение имени файла
   Filename = Application.GetSaveAsFilename( _
      InitialFileName:="myrange.xml", _
      fileFilter:="XML Files(*.xml), *.xml")
   If Filename = False Then Exit Sub

'  Открытие текстового файла
   Open Filename For Output As #1

'  Запись тегов <xml>
   Print #1, "<?xml version=""1.0"" encoding= _
      ""UTF-8"" standalone="" yes""?>"
   Print #1, "<EmployeeList
   xmlns:xsi=""http://www.w3.org/2001/XMLSchema-instance"">"

'  Циклический обход ячеек
   For r = 2 To Rng.Rows.Count
      Print #1, "<Employee>"
      For c = 1 To Rng.Columns.Count
         Print #1, "<" & Rng.Cells(1, c) & ">";
         If IsDate(Rng.Cells(r, c)) Then
            Print #1, Format(Rng.Cells(r, c), "yyyy-mm-dd");
         Else
            Print #1, Rng.Cells(r, c).Text;
         End If
         Print #1, "</" & Rng.Cells(1, c) & ">"
      Next c
      Print #1, "</Employee>"
   Next r
'  Закрытие таблицы
   Print #1, "</EmployeeList>"
'  Закрытие файла
   Close #1

'  Сообщение для пользователя
   MsgBox Rng.Rows.Count - 1 & " записей экспортировано в " _
       & Filename
End Sub
```

Компакт-диск

Этот пример находится на прилагаемом компакт-диске в файле export to XML.xlsm.

Экспортированный XML-файл можно открыть в Excel. После этого появится диалоговое окно, показанное на рис. 27.10. Если в этом окне установить переключатель XML-таблица, файл будет отображен в виде таблицы. При этом не сохраняются формулы, находящиеся в исходной таблице.

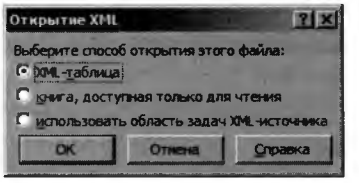


Рис. 27.10. После открытия XML-файла Excel предлагает три опции

Архивирование и разархивирование файлов

Возможно, наиболее часто для сжатия файлов используется формат ZIP. Даже файлы Excel 2010 сохраняются в формате ZIP (хотя и не используется расширение `.zip`). ZIP-файл может включать любое число файлов, а также целые папки. Содержимое этих файлов определяет степень сжатия при архивировании. Например, файлы формата JPG уже сжаты, поэтому сжатие этих файлов не приведет к заметной экономии места на диске.

Компакт-диск

Рассматриваемые в этом разделе примеры находятся на прилагаемом к книге компакт-диске в файлах `zip files.xlsm` и `unzip a file.xlsm`.

Архивирование файлов

В примере этого раздела демонстрируется способ создания ZIP-файла на основе группы выделенных пользователем файлов. Процедура `ZipFiles` отображает диалоговое окно, в котором пользователь может выбрать файлы. Затем создается ZIP-файл `compressed.zip` в папке Excel, заданной по умолчанию.

```
Sub ZipFiles()
   Dim ShellApp As Object
   Dim FileNameZip As Variant
   Dim FileNames As Variant
   Dim i As Long, FileCount As Long
'  Получить имена файлов
   FileNames = Application.GetOpenFilename _
      (FileFilter:="All Files (*.*),*.*", _
      FilterIndex:=1, _
      Title:="Выбор файлов для архивирования ", _
      MultiSelect:=True)
'  Выход при отмене отображения диалогового окна
   If Not IsArray(FileNames) Then Exit Sub
      FileCount = UBound(FileNames)
      FileNameZip = Application.DefaultFilePath _
         & "\compressed.zip"

'  Создание пустого Zip-файла с заголовком zip
   Open FileNameZip For Output As #1
   Print #1, Chr$(80) & Chr$(75) & Chr$(5) & Chr$(6) & _
```

```
      String(18, 0)
   Close #1
   Set ShellApp = CreateObject("Shell.Application")

'  Копирование файлов в сжатую папку
   For i = LBound(FileNames) To UBound(FileNames)
      ShellApp.Namespace(FileNameZip).CopyHere FileNames(i)
   Next i

'  Ожидание завершения выполнения сценария
   On Error Resume Next
   Do Until ShellApp.Namespace(FileNameZip).items.Count = _
         FileCount
      Application.Wait (Now + TimeValue("0:00:01"))
   Loop
   If MsgBox(FileCount & " Файлы архивированы в:" & _
      vbNewLine & FileNameZip & vbNewLine & vbNewLine & _
      "Просмотреть zip-файл?", vbQuestion + vbYesNo) = _
      vbYes Then Shell "Explorer.exe /e," & _
      FileNameZip, vbNormalFocus
End Sub
```

На рис. 27.11 показано диалоговое окно выбора файлов, которое сгенерировано с помощью метода `GetOpenFilename` объекта `Application` (дополнительные сведения об этом методе можно найти в главе 12). В этом диалоговом окне можно выбрать несколько файлов в одной папке.

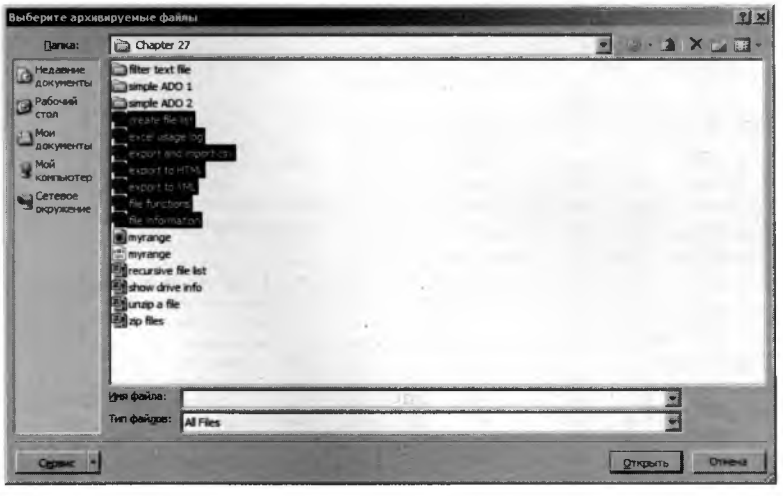


Рис. 27.11. В этом диалоговом окне выбираются архивируемые файлы

Процедура `ZipFiles` создает файл `compressed.zip` и записывает строку символов, которая идентифицирует его в качестве ZIP-файла. Затем создается объект `Shell.Application`, и метод `CopyHere` копирует файлы в ZIP-архив. После этого цикл `Do Until` каждую секунду подсчитывает файлы в ZIP-архиве. Это связано с тем, что копирование файлов требует времени, и если процедура завершает свою работу до окончания копирования файлов, ZIP-файл будет незавершен (и, возможно, поврежден).

Если количество файлов в ZIP-архиве соответствует заданной величине, выполнение цикла завершается и отображается сообщение для пользователя, показанное на рис. 27.12.

После щелчка на кнопке Да (Yes) открывается окно Проводника Windows, в котором показаны заархивированные файлы.

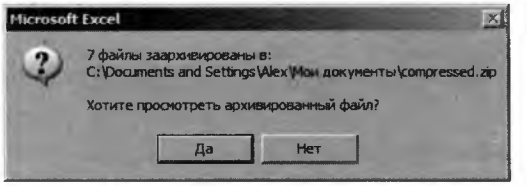


Рис. 27.12. После завершения создания ZIP-файла отображается сообщение для пользователя

Предупреждение

Процедура `ZipFiles`, представленная в этом разделе, упрощена в целях упрощения ее понимания. В частности, не выполняется проверка на наличие ошибок, а также отсутствует ряд других функций, например нельзя выбрать имя либо расположение ZIP-файла, а созданный ранее файл `compressed.zip` перезаписывается без предупреждения.

Разархивирование файла

В примере этого раздела рассматривается функция, противоположная функции из предыдущего раздела, — разархивирование. У пользователя запрашивается имя ZIP-файла, а затем разархивируются файлы, которые копируются в папку `Unzipped`, находящуюся в файловой папке Excel, заданной по умолчанию.

```
Sub UnzipAFile()
   Dim ShellApp As Object
   Dim TargetFile
   Dim ZipFolder
'  Целевой файл и временная папка
   TargetFile = Application.GetOpenFilename _
      (FileFilter:="Zip Files (*.zip), *.zip")
   If TargetFile = False Then Exit Sub
      ZipFolder = Application.DefaultFilePath & "\Unzipped\"
'  Создание временной папки
   On Error Resume Next
   RmDir ZipFolder
   MkDir ZipFolder
   On Error GoTo 0
'  Копирование архивированных файлов в новую папку
   Set ShellApp = CreateObject("Shell.Application")
   ShellApp.Namespace(ZipFolder).CopyHere _
      ShellApp.Namespace(TargetFile).items
   If MsgBox("Файлы разархивированы:" & _
      vbNewLine & ZipFolder & vbNewLine & vbNewLine & _
      "Просмотреть папку?", vbQuestion + vbYesNo) = vbYes Then _
         Shell "Explorer.exe /e," & ZipFolder, vbNormalFocus
End Sub
```

Процедура `UnzipAFile` использует метод `GetOpenFilename` для получения доступа к ZIP-файлу. Затем создается новая папка и используется объект `Shell.Application` для копирования содержимого ZIP-файла в новую папку. Затем пользователь выбирает отображение содержимого новой папки.

Модель ADO

ADO (ActiveX Data Objects — объекты данных AciveX) — это объектная модель, с помощью которой обеспечивается доступ к данным, сохраненным с применением самых различных форматов (включая общие форматы баз данных и даже текстовые файлы). Обратите внимание на то, что эта технология позволяет использовать единственную объектную модель для всех источников данных. В настоящее время ADO является одним из наиболее распространенных средств доступа к данным. Не путайте ADO с DAO (Data Access Objects — объекты доступа к данным).

В этом разделе рассматривается простой пример применения ADO для выборки записей из базы данных Access.

Примечание

Программирование с помощью ADO весьма сложно. Если вам необходимо получить доступ к внешним данным из Excel с помощью этой технологии, обратитесь к специальной литературе.

В примере кода `ADO_Demo` осуществляется выборка записей из базы данных Access `budget data.accdb`. Эта база включает одну таблицу (`Budget`). В этом примере выбираются данные из поля `Item`, в котором содержится текст `Lease`, из поля `Division`, в котором содержится текст `N. America`, а также из поля `Year`, в котором находится текст `2008`. Определяющие данные хранятся в объекте `Recordset`, причем эти данные будут переданы на рабочий лист (рис. 27.13).

	A	B	C	D	E	F	G	H	I	J	K	L
1	КОД	СОРТИР	ОТДЕЛЕ	ДЕПАРТАМЕ	КАТЕГОРИ	ЭЛЕМЕНТ	ГОД	МЕСЯЦ	ПЛАН	ФАКТ	ОТКЛОНЕНИЕ	
2	10	10	N. Amer	Data Proces	Facility	Lease	2008	Jan	3450	2631	819	
3	34	34	N. Amer	Human Resc	Facility	Lease	2008	Jan	4353	3875	478	
4	58	58	N. Amer	Accounting	Facility	Lease	2008	Jan	3898	2979	919	
5	82	82	N. Amer	Training	Facility	Lease	2008	Jan	3185	3545	-360	
6	106	106	N. Amer	Security	Facility	Lease	2008	Jan	3368	4120	-752	
7	130	130	N. Amer	R&D	Facility	Lease	2008	Jan	3926	3432	494	
8	154	154	N. Amer	Operations	Facility	Lease	2008	Jan	3329	3715	-386	
9	178	178	N. Amer	Shipping	Facility	Lease	2008	Jan	4095	2892	1203	
10	202	202	N. Amer	Sales	Facility	Lease	2008	Jan	3242	2687	555	
11	226	226	N. Amer	Advertising	Facility	Lease	2008	Jan	3933	3580	353	
12	250	250	N. Amer	Public Relati	Facility	Lease	2008	Jan	4316	4328	-12	
13	1330	1330	N. Amer	Data Proces	Facility	Lease	2008	Feb	4440	4357	83	
14	1354	1354	N. Amer	Human Resc	Facility	Lease	2008	Feb	4210	3196	1014	
15	1378	1378	N. Amer	Accounting	Facility	Lease	2008	Feb	2860	3658	-798	
16	1402	1402	N. Amer	Training	Facility	Lease	2008	Feb	4468	3759	709	
17	1426	1426	N. Amer	Security	Facility	Lease	2008	Feb	3499	3568	-69	
18	1450	1450	N. Amer	R&D	Facility	Lease	2008	Feb	3394	4196	-802	
19	1474	1474	N. Amer	Operations	Facility	Lease	2008	Feb	4187	3074	1113	
20	1498	1498	N. Amer	Shipping	Facility	Lease	2008	Feb	2870	3751	-881	
21	1522	1522	N. Amer	Sales	Facility	Lease	2008	Feb	4046	4013	33	

Рис. 27.13. Эта информация была получена из базы данных Access

```
Sub ADO_Demo()
'  Для выполнения этой процедуры установите ссылку на
'  библиотеку Microsoft ActiveX Data Objects 2.x Library
   Dim DBFullName As String
   Dim Cnct As String, Src As String
   Dim Connection As ADODB.Connection
   Dim Recordset As ADODB.Recordset
   Dim Col As Integer
   Cells.Clear
```

```
'  Информация о базе данных
   DBFullName = ThisWorkbook.Path & "\budget data.accdb"

'  Открытие соединения
   Set Connection = New ADODB.Connection
   Cnct = "Provider=Microsoft.ACE.OLEDB.12.0;"
   Cnct = Cnct & "Data Source=" & DBFullName & ";"
   Connection.Open ConnectionString:=Cnct

'  Создание набора записей
   Set Recordset = New ADODB.Recordset
   With Recordset
'     Фильтр
      Src = "SELECT * FROM Budget WHERE Item = 'Lease' "
      Src = Src & "and Division = 'N. America' "
      Src = Src & "and Year = '2008'"
      .Open Source:=Src, ActiveConnection:=Connection
'     Запись имени поля
      For Col = 0 To Recordset.Fields.Count - 1
         Range("A1").Offset(0, Col).Value = _
            Recordset.Fields(Col).Name
      Next
'     Запись набора записей
      Range("A1").Offset(1, 0).CopyFromRecordset Recordset
   End With
   Set Recordset = Nothing
   Connection.Close
   Set Connection = Nothing
End Sub
```

Компакт-диск

Рассматриваемый в этом разделе пример (`simple ado example.xlsm`) наряду с файлом базы данных Access (`budget data.accdb`) находится на прилагаемом к книге компакт-диске. Также на этом компакт-диске можно найти пример использования технологии ADO для запроса текстового файла CSV. Файл `simple ado example2.xlsm` использует большой CSV-файл под именем `music_list.csv`.

Глава 28

Управление компонентами Visual Basic

В этой главе...

- Введение в IDE
- Объектная модель IDE
- Отображение всех компонентов проекта VBA
- Отображение всех процедур VBA, содержащихся в рабочей книге
- Замещение модуля обновленной версией
- Использование VBA для создания кода VBA
- Добавление элементов управления в диалоговое окно UserForm на этапе разработки
- Программное создание диалоговых окон UserForm

В данной главе углубленно рассматриваются методы управления компонентами VBE (среда разработки кода VBA).

Введение в IDE

В данной главе рассматривается тема, которая может оказаться довольно важной для многих читателей, — создание кода VBA для управления компонентами проектов. Интегрированная среда разработки VBA включает объектную модель, которая содержит ключевые элементы проекта VBA, включая сам редактор. В результате можно создать код VBA, который добавляет или удаляет модули, создает дополнительный код VBA или даже диалоговые окна `UserForm`.

IDE — это интегрированная среда разработки для редактора Visual Basic Editor с поддержкой интерфейса автоматизации OLE. Сразу после создания ссылки на библиотеку Visual Basic Extensibility Library разработчику предоставляется доступ ко всем объ-

ектам, свойствам и методам VBE, а также разрешается объявлять объекты из классов-членов IDE.

После выполнения команды VBE Tools⇨References (Сервис⇨Ссылки) отображается диалоговое окно References (Ссылки), с помощью которого можно добавлять ссылки в библиотеку Microsoft Visual Basic for Applications Extensibility Library (рис. 28.1). В результате предоставляется доступ к объекту `VBIDE`. Создание ссылки на объект `VBIDE` позволит объявлять объекты, которые входят в состав `VBIDE`, а также открывает доступ к константам, определенным внутри IDE. Отметим, что доступ к объектам IDE можно получать, даже не создавая ссылку, но при этом у вас не будет возможности использовать константы IDE, а также объявлять объекты, которые ссылаются на компоненты IDE.

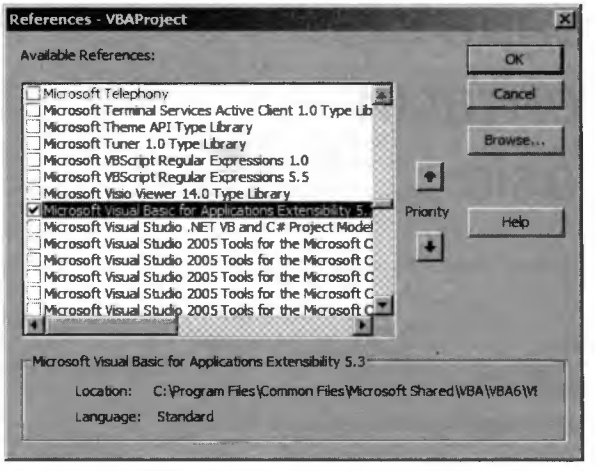


Рис. 28.1. Добавление ссылки на библиотеку Microsoft Visual Basic for Applications Extensibility Library

Перекрестная ссылка

Дополнительные сведения относительно автоматизации OLE можно найти в главе 20.

Получив предоставление о работе объектной модели IDE, можете приступать к созданию кода, который будет выполнять ряд полезных функций:

- создавать и удалять модули VBA;
- вставлять код VBA;
- создавать пользовательские диалоговые окна;
- добавлять элементы управления в диалоговые окна `UserForm`.

О системе безопасности Excel

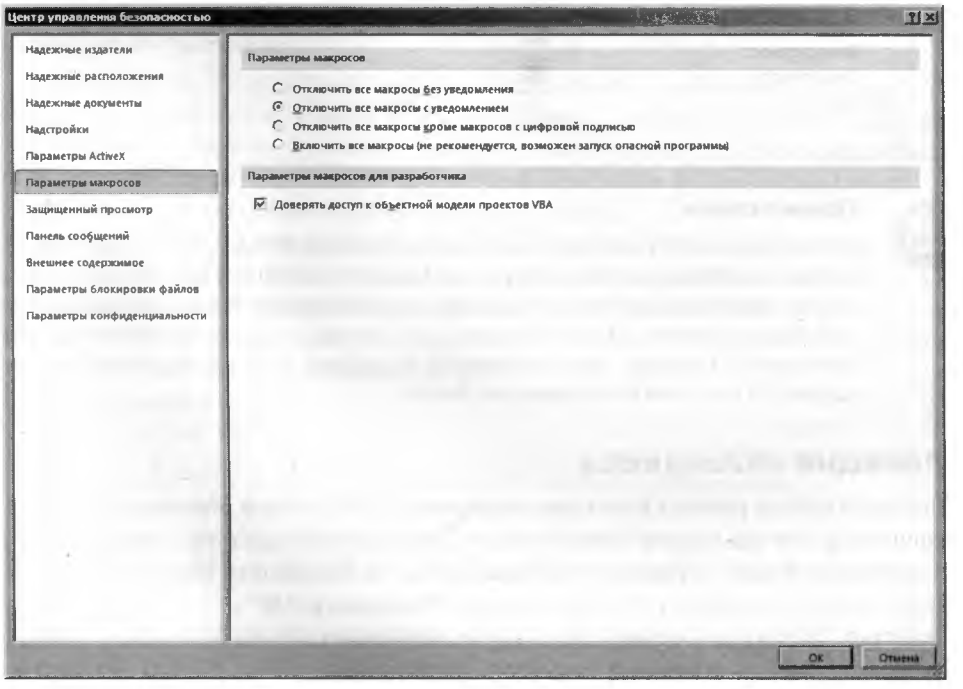
Если вы разрабатываете процедуры VBA для их использования другими пользователями Excel, учтите, что примеры кода из этой главы могут не работать на компьютерах пользователей. Причина этого явления — засилье компьютерных вирусов. Поэтому специалисты из компании Microsoft (начиная с версии Excel 2002) существенно затруднили изменение компонентов проекта VBA с помощью макросов VBA. Если вы попытаетесь выполнить любую из процедур, код которых приведен в данной главе, на экране, скорее всего, появится сообщение об ошибке.

Вероятность появления сообщения об ошибке зависит от настроек, выбранных в диалоговом окне Центр управления безопасностью (Trust Center). Для просмотра либо изменения этих настроек выполните следующие действия:

- выберите команду Файл⇨Параметры Excel (File⇨Excel Options);
- в диалоговом окне Параметры Excel выберите раздел Центр управления безопасностью (Trust Center);
- в разделе Центр управления безопасностью щелкните на кнопке Параметры центра управления безопасностью (Trust Center Settings);
- в диалоговом окне Центр управления безопасностью (Trust Center) выберите раздел Параметры макросов (Macro Settings).

Для быстрого перехода к этому диалоговому окну выполните команду Разработчик⇨Код⇨Безопасность макросов (Developer⇨Code⇨Macro Security).

В этом окне находится флажок Доверять доступ к объектой модели проектов VBA (Trust Access to the VBA Project Object Model).



Этот флажок по умолчанию сброшен. Если даже пользователь выберет параметр, активизирующий макросы рабочей книги, без установки этого флажка изменение проекта VBA невозможно. Обратите внимание на то, что эта настройка применяется ко всем рабочим книгам одновременно, а не к какой-то определенной книге.

Состояние этого флажка невозможно определить с помощью кода VBA. Существует косвенный метод, который предусматривает обращение к объекту `VBProject` с последующей проверкой наличия ошибки. Для этого применяется следующий код.

```
On Error Resume Next
   Set x = ActiveWorkbook.VBProject
   If Err <> 0 Then
      MsgBox "Настройки безопасности блокируют этот макрос."
```

```
        Exit Sub
End If
```

Обратите внимание: далеко не все примеры предназначены для обычных пользователей Excel. Многие из них могут применяться разработчиками, создающими программные проекты. Если вы являетесь одним из них, установите флажок Доверять доступ к объектой модели проектов VBA (Trust Access to Visual Basic Project).

Объектная модель IDE

Программирование IDE требует четкого понимания объектной модели. Объектом верхнего уровня в иерархии объектов выступает VBE (Visual Basic Environment — среда разработки Visual Basic). Как и в случае с объектной моделью Excel, VBE содержит другие объекты. Упрощенная версия объектной иерархии IDE выглядит следующим образом.

```
VBE
   VBProject
      VBComponent
         CodeModule
         Designer
         Property
      Reference
   Window
   CommandBar
```

Примечание

В этой главе отсутствуют описания коллекций `Windows` и `CommandBars` из библиотеки Microsoft Visual Basic for Applications Extensibility Library, поскольку они практически бесполезны для разработчиков Excel. Основное внимание будет уделено объекту `VBProject`, который часто применяется разработчиками. Прежде чем приступить к работе с этим объектом, прочтите врезку "О системе безопасности Excel".

Коллекция `VBProjects`

Каждая открытая рабочая книга или надстройка представлена объектом `VBProject`, входящим в состав коллекции `VBProjects`. Для получения доступа к объекту `VBProject` для рабочей книги проверьте наличие ссылки на библиотеку Microsoft Visual Basic for Applications Extensibility Library (см. раздел "Введение в IDE").

Свойство `VBProject` объекта `Workbook` возвращает объект `VBProject`. Следующий оператор создает переменную, которая представляет объект `VBProject` активной рабочей книги.

```
Dim VBP As VBProject
Set VBP = ActiveWorkbook.VBProject
```

Примечание

Если при выполнении оператора `Dim` выводится сообщение об ошибке, то необходимо проверить существование ссылки на библиотеку Microsoft Visual Basic for Application Extensibility.

Каждый объект `VBProject` содержит коллекцию компонентов VBA, которые входят в проект (диалоговые окна `UserForm`, модули кода, модули классов, а также модули до-

кументов). Данная коллекция называется VBComponents. Кроме того, объект VBProject содержит коллекцию References текущего проекта, представляющую библиотеки, на которые ссылается этот проект.

Невозможно непосредственно добавить новый элемент в коллекцию VBProjects. Данная задача выполняется в процессе создания или открытия новой рабочей книги Excel. При этом автоматически добавляется новый элемент в коллекцию VBProjects. Также невозможно непосредственно удалить объект VBProject; для удаления объекта VBProject из коллекции нужно закрыть рабочую книгу.

Коллекция VBComponents

Для того чтобы получить доступ к члену коллекции VBComponents, необходимо воспользоваться свойством VBComponents с указанием индексного номера или имени. Следующие операторы демонстрируют два способа получения доступа к компонентам VBA.

```
Set VBC = ThisWorkbook.VBProject.VBComponents(1)
Set VBC = ThisWorkbook.VBProject.VBComponents("Module1")
```

Коллекция References

Каждый проект VBA в Excel содержит определенное количество ссылок. Ссылки проекта можно добавлять, удалять и редактировать с помощью команды Tools⇨References (Сервис⇨Ссылки) (рис. 28.1). Каждый проект содержит ссылки на библиотеки объектов (например, VBA, Excel, OLE Automation, а также Office Object Library), а при необходимости в проект можно добавить новые ссылки.

Ссылками в проекте можно также управлять с помощью кода VBA. Коллекция References содержит объекты Reference, класс которых предоставляет все необходимые свойства и методы. Приведенная далее процедура отображает окно сообщения, которое содержит значения свойств Name, Description и FullPath каждого объекта Reference проекта активной рабочей книги.

```
Sub ListReferences()
   Dim Ref As Reference
   Msg = ""
      For Each Ref In ActiveWorkbook.VBProject.References
         Msg = Msg & Ref.Name & vbNewLine
         Msg = Msg & Ref.Description & vbNewLine
         Msg = Msg & Ref.FullPath & vbNewLine & vbNewLine
      Next Ref
   MsgBox Msg
End Sub
```

На рис. 28.2 показан результат выполнения этой процедуры для рабочей книги, содержащей шесть активных ссылок.

Примечание

Поскольку объектная переменная имеет тип Reference, процедура ListReferences требует установки ссылки на библиотеку VBA Extensibility Library. Если объявить переменную Ref как имеющую универсальный тип Object, ссылка на библиотеку VBA Extensibility Library не требуется.

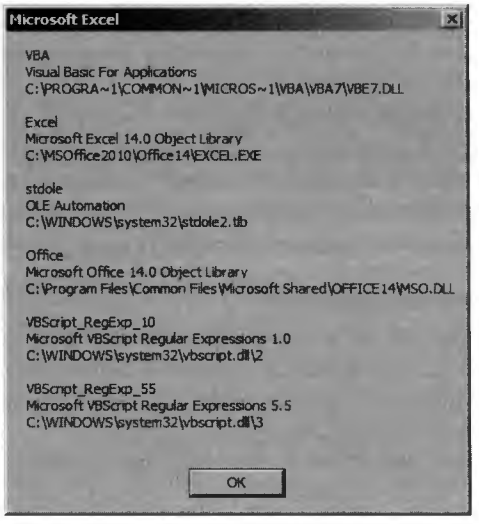


Рис. 28.2. В этом окне сообщения отображается информация о ссылках, входящих в состав проекта

Ссылки можно добавлять программно. Для этого используется один из двух методов класса `Reference`. Метод `AddFromFile` добавляет ссылку, если известно имя файла и путь к папке, в которой он хранится. Метод `AddFromGuid` добавляет ссылку, если известно значение *глобально уникального идентификатора*, или GUID. Дополнительная информация по этой теме приводится в справочном руководстве.

Отображение всех компонентов проекта VBA

Процедура `ShowComponents`, код которой приведен ниже, просматривает каждый компонент VBA в активной рабочей книге и записывает следующую информацию на рабочий лист:

- название компонента;
- тип компонента;
- количество строк кода в модуле кода этого компонента.

```
Sub ShowComponents()
   Dim VBP As VBIDE.VBProject
   Dim VBC As VBComponent
   Dim row As Long

   Set VBP = ActiveWorkbook.VBProject
'  Заголовки
   Cells.ClearContents
   Range("A1:C1") = Array("Имя", "Тип", "Строки кода")
   Range("A1:C1").Font.Bold = True
   row = 1
'  Циклический просмотр компонентов VB
   For Each VBC In VBP.VBComponents
      row = row + 1
'     Название
```

```
            Cells(row, 1) = VBC.Name
'           Тип
            Select Case VBC.Type
               Case vbext_ct_StdModule
                  Cells(row, 2) = "Модуль"
               Case vbext_ct_ClassModule
                  Cells(row, 2) = "Модуль класса"
               Case vbext_ct_MSForm
                  Cells(row, 2) = "Пользовательская форма"
               Case vbext_ct_Document
                  Cells(row, 2) = "Модуль документа"
            End Select
'           Строки кода
            Cells(row, 3) = VBC.CodeModule.CountOfLines
         Next VBC
      End Sub
```

Обратите внимание на то, что в коде использовались встроенные константы (например, `vbext_ct_StdModule`) для определения типа компонента. Эти константы не определяются до тех пор, пока не будет установлена ссылка на библиотеку Microsoft Visual Basic for Applications Extensibility Library.

На рис. 28.3 показан результат выполнения процедуры `ShowComponents`. В этом случае проект VBA содержит семь компонентов, и только два из них включают непустой модуль кода.

	A	B	C	D
1	Имя	Тип	Строки кода	
2	ThisWorkbook	Модуль документа	0	
3	Sheet1	Модуль документа	0	
4	Module1	Модуль	50	
5	Class1	Модуль класса	0	
6	UserForm1	Форма UserForm	46	
7	Sheet2	Модуль документа	0	
8	Sheet3	Модуль документа	0	
9				

Рис. 28.3. Так выглядит результат выполнения процедуры `ShowComponents`

Компакт-диск

Рассматриваемая в этом разделе процедура находится на прилагаемом к книге компакт-диске в рабочей книге `list VB components.xlsm`. Обратите внимание на то, что она включает ссылку на библиотеку VBA Extensibility Library, которая позволяет отобразить все открытые VB-проекты.

Отображение всех процедур VBA, содержащихся в рабочей книге

Макрос `ListProcedures`, рассматриваемый в этом разделе, создает список всех процедур VBA, находящихся в активной рабочей книге, и отображает его в окне сообщения.

```
Sub ListProcedures()
   Dim VBP As VBIDE.VBProject
   Dim VBC As VBComponent
   Dim CM As CodeModule
```

```
    Dim StartLine As Long
    Dim Msg As String
    Dim ProcName As String

'   Используется активная рабочая книга
    Set VBP = ActiveWorkbook.VBProject

'   Циклический обход процедур VB
    For Each VBC In VBP.VBComponents
        Set CM = VBC.CodeModule
        Msg = Msg & vbNewLine
        StartLine = CM.CountOfDeclarationLines + 1
        Do Until StartLine >= CM.CountOfLines
            Msg = Msg & VBC.Name & ": " & _
                CM.ProcOfLine(StartLine, vbext_pk_Proc) & vbNewLine
            StartLine = StartLine + CM.ProcCountLines _
                (CM.ProcOfLine(StartLine, vbext_pk_Proc), _
            vbext_pk_Proc)
        Loop
    Next VBC
    MsgBox Msg
End Sub
```

На рис. 28.4 показан результат выполнения этой процедуры в случае, когда рабочая книга содержит девять процедур.

Компакт-диск

Рассматриваемый в этом разделе пример находится на прилагаемом компакт-диске в файле `list all procedures.xlsm`.

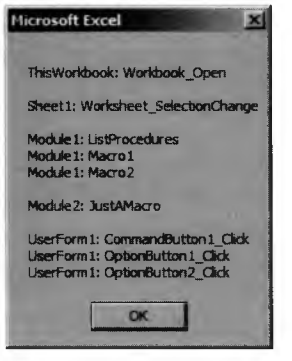


Рис. 28.4. В окне сообщения отображаются все процедуры, находящиеся в активной рабочей книге

Замещение модуля обновленной версией

Пример, приведенный в этом разделе, демонстрирует замещение модуля VBA другим модулем VBA. Кроме демонстрации трех методов объекта `VBComponent` (`Export`, `Remove` и `Import`), процедура имеет и практическое применение. Например, после того как рабочая книга была распространена среди группы пользователей, выяснилось, что в макросе допущена ошибка, поэтому он требует обновления. Поскольку у пользователей была возможность добавлять данные в рабочую книгу, ее полное замещение могло оказаться нецелесообразным. Решением в такой ситуации служит распространение другой рабочей книги, которая содержит макрос, замещающий модуль VBA обновленной версией, хранящейся в файле.

Этот пример включает две рабочие книги:

- `UserBook.xlsm` — эта рабочая книга содержит модуль (`Module1`), который необходимо заместить;
- `UpdateUserBook.xlsm` — содержит процедуру VBA, замещающую модуль `Module1` в рабочей книге `UserBook.xlsm` на обновленную версию.

Ниже представлен код процедуры BeginUpdate. Этот макрос входит в состав рабочей книги UpdateUserBook.xlsm, которую следует передать пользователям книги UserBook.xlsm. В процессе выполнения этой процедуры проверяется, открыта ли книга UserBook.xlsm. Затем отображается сообщение для пользователей, показанное на рис. 28.5.

Рис. 28.5. В этом окне появляется сообщение о том, что модуль будет обновлен

```
Sub BeginUpdate()
   Dim Filename As String
   Dim Msg As String
   Filename = "UserBook.xlsm"

'  Активизация рабочей книги
   On Error Resume Next
   Workbooks(Filename).Activate
   If Err <> 0 Then
      MsgBox Filename & " должен быть открыт.", vbCritical
      Exit Sub
   End If
   Msg = "Этот макрос замещает Module1 в UserBook.xlsm "
   Msg = Msg & "обновленной версией." & vbCrLf & vbCrLf
   Msg = Msg & "Щелкните на кнопке OK для продолжения."
   If MsgBox(Msg, vbInformation + vbOKCancel) = vbOK Then
      Call ReplaceModule
   Else
      MsgBox "Модуль не заменен,", vbCritical
   End If
End Sub
```

После щелчка на кнопке OK (для подтверждения замены) вызывается процедура ReplaceModule. Эта процедура заменяет существующий модуль Module1 в рабочей книге UserBook.xlsm копией модуля Module1, находящейся в файле UpdateUserBook.xlsm.

```
Sub ReplaceModule()
   Dim ModuleFile As String
   Dim VBP As VBIDE.VBProject
'  Экспортирование модуля Module1 из этой книги
   ModuleFile = Application.DefaultFilePath & "\tempmodxxx.bas"
   ThisWorkbook.VBProject.VBComponents("Module1") _
      .Export ModuleFile

'  Замена модуля Module1 в книге UserBook
   Set VBP = Workbooks("UserBook.xlsm").VBProject
   On Error GoTo ErrHandle
   With VBP.VBComponents
      .Remove VBP.VBComponents("Module1")
      .Import ModuleFile
```

```
    End With

'   Удаление временного файла модуля
    Kill ModuleFile
    MsgBox "Модуль заменен.", vbInformation
    Exit Sub
ErrHandle:
'   Произошла ошибка?
    MsgBox "ОШИБКА. Модуль не может быть заменен.", _
       vbCritical
End Sub
```

Описанная процедура выполняет следующие действия.

1. Сначала в файл экспортируется модуль `Module1` (обновленная версия). Файлу присваивается случайное имя, чтобы снизить вероятность перезаписи существующего файла.
2. Из рабочей книги `UserForm.xlsm` удаляется модуль `Module1` (обновляемая версия). Для этого используется метод `Remove` из коллекции `VBComponents`.
3. После этого процедура импортирует модуль (сохраненный на первом этапе) в рабочую книгу `UserBook.xlsm`.
4. Удаляется временный файл с обновленной версией модуля.
5. Выводится окно с сообщением, которое содержит информацию о результате обновления. Обработка ошибок необходима для своевременного извещения пользователя о возникновении ошибки.

Компакт-диск

Рассматриваемый в этом разделе пример находится на прилагаемом к книге компакт-диске в двух файлах: `UserBook.xlsm` и `UpdateUserBook.xlsm`.

Использование VBA для создания кода VBA

Пример, приведенный в этом разделе, демонстрирует использование VBA для создания дополнительного кода VBA. Процедура `AddButtonAndCode` выполняет следующие действия.

1. Добавляет новый рабочий лист.
2. Добавляет на рабочий лист элемент управления `CommandButton`.
3. Меняет расположение, размер и название элемента управления `CommandButton`.
4. Добавляет процедуру обработки события для элемента управления `CommandButton`.

   Такая процедура называется `CommandButton1_Click`. Она располагается в модуле кода рабочего листа и используется для активизации рабочего листа `Лист1`.

Ниже приводится код процедуры `AddButtonAndCode`.

```
Sub AddButtonAndCode()
    Dim NewSheet As Worksheet
    Dim NewButton As OLEObject
```

```
'  Добавление листа
   Set NewSheet = Sheets.Add

'  Добавление кнопки
   Set NewButton = NewSheet.OLEObjects.Add _
      ("Forms.CommandButton.1")
   With NewButton
      .Left = 4
      .Top = 4
      .Width = 100
      .Height = 24
      .Object.Caption = "Возврат на Лист1"
   End With

'  Код обработчика событий
   Code = "Sub CommandButton1_Click()" & vbCrLf
   Code = Code & " On Error Resume Next" & vbCrLf
   Code = Code & " Sheets(""Лист1"").Activate" & vbCrLf
   Code = Code & " If Err <> 0 Then" & vbCrLf
   Code = Code & " MsgBox ""Невозможно активизировать Лист1."""  _
      & vbCrLf
   Code = Code & " End If" & vbCrLf
   Code = Code & "End Sub"

   With ActiveWorkbook.VBProject. _
      VBComponents(NewSheet.Name).CodeModule
         NextLine = .CountOfLines + 1
            .InsertLines NextLine, Code
   End With
End Sub
```

На рис. 28.6 показаны рабочий лист и элемент управления CommandButton, которые были добавлены с помощью процедуры AddButtonAndCode.

Рис. 28.6. Этот лист, элемент управления CommandButton и обработчик событий были добавлены с помощью кода VBA

Компакт-диск

Рассматриваемый в этом разделе пример находится на прилагаемом к книге компакт-диске в файле add button and code.xlsm.

Самой сложной задачей процедуры является добавление кода VBA в модуль кода нового рабочего листа. Код хранится в переменной, которая называется Code. Каждый оператор отделен последовательностью символов возврата каретки и перевода строки. Метод InsertLines добавляет код в модуль кода добавленного рабочего листа.

Переменная NextLine хранит количество существующих строк в модуле кода и каждый раз при добавлении строки увеличивает свое значение на единицу. Таким образом обеспечивается вставка кода в конец модуля. Если вставлять код начиная с первой строки, а система пользователя настроена на автоматическое добавление оператора Option Explicit в модуль, то возникнет сообщение об ошибке.

На рис. 28.7 показана процедура, которая создана в результате выполнения процедуры AddButtonAndCode.

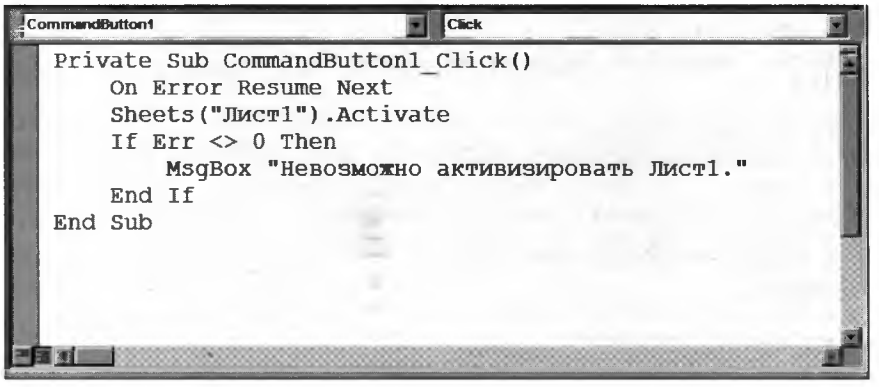


Рис. 28.7. Этот код VBA был сгенерирован процедурой обработки событий

Добавление элементов управления в диалоговое окно UserForm на этапе разработки

Тот, кто потратил не одну бессонную ночь на создание диалоговых окон UserForm, знает, что данная задача может быть утомительной, поскольку после добавления элемента управления необходимо изменить его размер и расположение, чтобы правильно выровнять его относительно остальных элементов управления в диалоговом окне. Даже если полностью использовать потенциал команд форматирования VBE, настройка внешнего вида элементов управления может занять немало времени.

Диалоговое окно UserForm, которое показано на рис. 28.8 (под названием UserForm1), содержит 100 элементов управления CommandButton. Все они имеют одинаковый размер и выровнены в диалоговом окне. Кроме того, каждый элемент управления CommandButton имеет собственную процедуру обработки события. Добавление элементов управления вручную и создание для них процедур обработки событий может занять довольно много времени — точнее, почти бесконечность. Автоматическое добавление этих элементов управления на этапе разработки с помощью кода VBA займет всего несколько секунд.

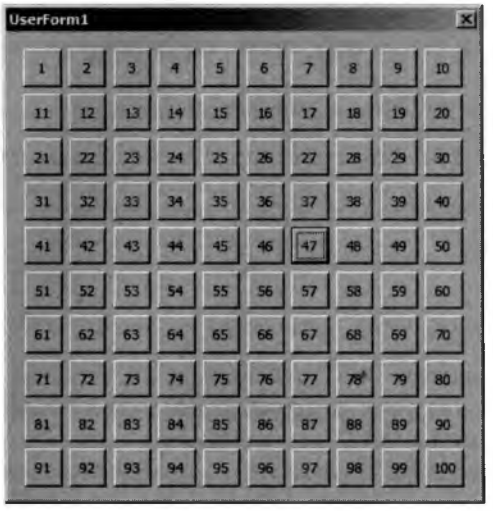


Рис. 28.8. Для добавления элементов управления `CommandButton` в диалоговое окно `UserForm` используется процедура VBA

Управление диалоговыми окнами `UserForm` на этапе разработки и этапе выполнения

Важно понимать разницу между управлением диалоговым окном `UserForm` и его элементами управления на этапе разработки и на этапе выполнения. Управление диалоговым окном `UserForm` на этапе выполнения становится заметно для пользователя, поскольку диалоговое окно отображается на экране. Но изменения, внесенные на этапе выполнения, непостоянные. Если создать код, который на этапе выполнения будет изменять значение свойства `Caption` диалогового окна, то после отображения этого диалогового окна пользователь увидит новый заголовок. Если же после этого вернуться в редактор VBE, то окажется, что диалоговое окно `UserForm` имеет исходный заголовок. В части IV содержится ряд примеров, которые демонстрируют методы управления диалоговыми окнами `UserForm` и их элементами управления на этапе выполнения.

С другой стороны, управление на этапе разработки имеет постоянную природу. Оно является аналогом ручного изменения свойств элементов управления и диалоговых окон с помощью инструментов редактора VBE. Как правило, код VBA на этапе разработки используется для автоматизации операций над создаваемыми элементами управления и диалоговыми окнами `UserForm`. Для того чтобы изменить элементы управления на этапе разработки, необходимо получить доступ к объекту `Designer` диалогового окна `UserForm`.

Чтобы продемонстрировать разницу между управлением на этапе разработки и управлением на этапе выполнения, были созданы две простые процедуры, которые добавляют элемент управления `CommandButton` в диалоговое окно `UserForm`. Одна процедура добавляет элемент управления на этапе выполнения, а вторая — на этапе разработки.

Процедура `RunTimeButton`, приведенная ниже, очень проста. Она находится в модуле кода общего назначения и служит для создания элемента управления `CommandButton` и изменения значений его свойств. После добавления элемента управления отображается диалоговое окно `UserForm`. Элемент управления `CommandButton` можно увидеть в диа-

логовом окне в момент его появления на экране. Если просмотреть диалоговое окно в редакторе VBE, то окажется, что элемент управления на форме отсутствует.

```
Sub RunTimeButton()
'  Добавление кнопки на этапе выполнения
   Dim Butn As CommandButton
   Set Butn = UserForm1.Controls.Add("Forms.CommandButton.1")
   With Butn
      .Caption = "Добавлено на этапе выполнения "
      .Width = 100
      .Top = 10
   End With
   UserForm1.Show
End Sub
```

Ниже приведена процедура DesignTimeButton. Отличие заключается в том, что она использует объект Designer, который содержится в объекте VBComponent. В частности, применяется метод Add для добавления элемента управления CommandButton. Поскольку процедура обращается к объекту Designer, элемент управления CommandButton добавляется в диалоговое окно UserForm так же, как это выполняется при ручном редактировании диалогового окна.

```
Sub DesignTimeButton()
'  Добавление кнопки на этапе разработки
   Dim Butn As CommandButton
   Set Butn = ThisWorkbook.VBProject. _
      VBComponents("UserForm1") _
   .Designer.Controls.Add("Forms.CommandButton.1")
   With Butn
      .Caption = "Добавлено на этапе разработки"
      .Width = 120
      .Top = 40
   End With
End Sub
```

Добавление 100 элементов управления CommandButton на этапе разработки

Пример данного раздела демонстрирует преимущества использования объекта Designer при создании диалоговых окон UserForm. В этом случае код добавляет 100 элементов управления CommandButton (которые соответствующим образом расположены и выровнены), устанавливает значение свойства Caption для каждого элемента управления CommandButton и создает 100 процедур обработки событий (для каждого элемента управления CommandButton).

```
Sub Add100Buttons()
   Dim UFvbc As VBComponent
   Dim CMod As CodeModule
   Dim ctl As Control
   Dim cb As CommandButton
   Dim n As Long, c As Long, r As Long
   Dim code As String

   Set UFvbc = ThisWorkbook.VBProject.VBComponents("UserForm1")
'  Удаление всех элементов управления
   For Each ctl In UFvbc.Designer.Controls
```

```
      UFvbc.Designer.Controls.Remove ctl.Name
   Next ctl
'  Удаление всего кода VBA
   UFvbc.CodeModule.DeleteLines 1, UFvbc.CodeModule.CountOfLines
'  Добавление 100 элементов управления CommandButton
   n = 1
   For r = 1 To 10
      For c = 1 To 10
         Set cb = UFvbc.Designer. _
            Controls.Add("Forms.CommandButton.1")
         With cb
            .Width = 22
            .Height = 22
            .Left = (c * 26) - 16
            .Top = (r * 26) - 16
            .Caption = n
         End With
'        Добавление кода обработки событий
         With UFvbc.CodeModule
            code = ""
            code = code & "Private Sub CommandButton" & n & _
               "_Click" & vbCr
            code = code & "Msgbox ""Кнопка" & n & _
               """" & vbCr
            code = code & "End Sub"
               .InsertLines .CountOfLines + 1, code
            End With
            n = n + 1
         Next c
      Next r
End Sub
```

Компакт-диск

Рассматриваемый в этом разделе пример находится на прилагаемом к книге компакт-диске в файле `add 100 buttons.xlsm`.

Процедура `Add100Buttons` требует наличия диалогового окна `UserForm` (под названием `UserForm1`). Процедура начинает работу с удаления всех элементов управления в этом диалоговом окне. Затем удаляется весь код в модуле кода данного диалогового окна. Для этого используется метод `DeleteLines` объекта `CodeModule`. Далее добавляются элементы управления `CommandButton` и процедуры обработки события для этих элементов управления с помощью двух циклов `For-Next`. Процедуры обработки событий очень просты. Ниже приведен пример одной из них (процедура задана для элемента управления `CommandButton1`).

```
Private Sub CommandButton1_Click()
   MsgBox "Это CommandButton1"
End Sub
```

Если нужно отобразить на экране форму после добавления элементов управления на этапе разработки, добавьте следующую инструкцию непосредственно перед оператором `End Sub`.

```
VBA.UserForms.Add("UserForm1").Show
```

Вам потребуется немало времени, чтобы выяснить, как необходимо отображать диалоговое окно `UserForm`. Если интерпретатор VBA генерирует диалоговое окно `User-`

Form, в котором насчитывается 100 кнопок, то это диалоговое окно содержится в памяти интерпретатора, но оно не является частью проекта. Для того чтобы формально добавить диалоговое окно UserForm1 в коллекцию UserForms, необходимо воспользоваться методом Add. Возвращаемое этим методом значение (да, этот метод возвращает значение) является ссылкой на диалоговое окно. Именно поэтому в конце вызова метода Add необходимо вызвать метод Show. Из этого следует правило, согласно которому для отображения созданного диалогового окна его сначала необходимо добавить в коллекцию UserForms.

Программное создание диалоговых окон UserForm

В завершение данной главы речь пойдет об использовании средств VBA для создания диалоговых окон UserForm на этапе выполнения. В этом разделе представлены два примера: один из них довольно прост, второй — намного сложнее.

Простой пример

Пример, приведенный в этом разделе, не имеет особой практической ценности. Точнее, он совершенно бесполезен. Однако этот пример демонстрирует важные концепции. Процедура MakeForm выполняет следующие задачи.

1. Создает временное диалоговое окно UserForm в активной рабочей книге. Для этого используется метод Add коллекции VBComponents.
2. Добавляет элемент управления CommandButton в диалоговое окно UserForm. Для этого применяется объект Designer.
3. Добавляет процедуру обработки события в модуль кода диалогового окна UserForm.

   Данная процедура (CommandButton1_Click) используется для отображения окна сообщения и закрытия диалогового окна.
4. Отображает диалоговое окно UserForm.
5. Удаляет диалоговое окно UserForm.

Общим результатом работы процедуры является динамическое создание диалогового окна UserForm, его использование и удаление после использования. Приводимый пример, а также пример в следующем разделе стирают границу между модификацией диалоговых окон на этапе разработки и на этапе выполнения. Диалоговое окно создается с помощью методов разработки, но все это происходит на этапе выполнения.

Ниже приводится пример процедуры MakeForm.

```
Sub MakeForm()
   Dim TempForm As Object
   Dim NewButton As Msforms.CommandButton
   Dim Line As Integer
   Application.VBE.MainWindow.Visible = False
'  Создание диалогового окна UserForm
   Set TempForm = ThisWorkbook.VBProject. _
      VBComponents.Add(3) 'vbext_ct_MSForm
   With TempForm
      .Properties("Caption") = "Временная форма"
```

```
        .Properties("Width") = 200
        .Properties("Height") = 100
    End With
'   Добавление элемента управления CommandButton
    Set NewButton = TempForm.Designer.Controls _
        .Add("Forms.CommandButton.1")
    With NewButton
        .Caption = "Щелкни!"
        .Left = 60
        .Top = 40
    End With
'   Добавление обработчика событий для кнопки
    With TempForm.CodeModule
       Line = .CountOfLines
        .InsertLines Line + 1, "Sub CommandButton1_Click()"
        .InsertLines Line + 2, " MsgBox ""Привет!"""
        .InsertLines Line + 3, " Выгрузи меня"
        .InsertLines Line + 4, "End Sub"
    End With
'   Отображение формы
    VBA.UserForms.Add(TempForm.Name).Show
'
'   Удаление формы
    ThisWorkbook.VBProject.VBComponents.Remove TempForm
End Sub
```

Компакт-диск

Этот пример находится на прилагаемом к книге компакт-диске в файле `create userform on the fly.xlsm`.

Процедура `MakeForm` создает и отображает простое окно `UserForm`, показанное на рис. 28.9.

Рис. 28.9. Это окно `UserForm` и лежащий в его основании код были созданы за несколько минут

Примечание

Для рабочей книги, включающей процедуру `MakeForm`, не требуется ссылка на библиотеку VBA Extensibility Library. Это связано с тем, что временная форма `TempForm` объявляется как универсальный тип данных `Object` (а не в виде объекта `VBComponent`). Также не используется ни одна из встроенных констант.

Обратите внимание на то, что одна из первых инструкций скрывает окно VBE путем присваивания свойству `Visible` этого окна значения `False`. Это позволяет избежать мерцания экрана при генерации кода и диалогового окна.

Более сложный пример

Данный пример имеет как образовательную, так и практическую ценность. Он состоит из функции GetOption, которая отображает диалоговое окно UserForm. В этом диалоговом окне находится несколько элементов управления OptionButton, подписи которых указываются в виде аргументов функции. Функция возвращает значение, которое соответствует элементу управления OptionButton, выбранному пользователем.

Компакт-диск

Рассматриваемый в этом разделе пример находится на прилагаемом к книге компакт-диске в файле getoption function.xlsm.

Ниже приводится код функции GetOption.

```
Function GetOption(OpArray, Default, Title)
   Dim TempForm As Object
   Dim NewOptionButton As Msforms.OptionButton
   Dim NewCommandButton1 As Msforms.CommandButton
   Dim NewCommandButton2 As Msforms.CommandButton
   Dim i As Integer, TopPos As Integer
   Dim MaxWidth As Long
   Dim Code As String

'  Сокрытие окна VBE во избежание мерцания экрана
   Application.VBE.MainWindow.Visible = False
'   Создание окна UserForm
   Set TempForm = _
      ThisWorkbook.VBProject.VBComponents.Add(3)
   TempForm.Properties("Width") = 800

'  Добавление элементов управления OptionButton
   TopPos = 4
   MaxWidth = 0 ' Ширина наиболее широкой кнопки OptionButton
   For i = LBound(OpArray) To UBound(OpArray)
      Set NewOptionButton = TempForm.Designer.Controls. _
         Add("Forms.OptionButton.1")
      With NewOptionButton
         .Width = 800
         .Caption = OpArray(i)
         .Height = 15
         .Accelerator = Left(.Caption, 1)
         .Left = 8
         .Top = TopPos
         .Tag = i
         .AutoSize = True
         If Default = i Then .Value = True
         If .Width > MaxWidth Then MaxWidth = .Width
      End With
      TopPos = TopPos + 15
   Next i

'  Добавление кнопки Отмена
   Set NewCommandButton1 = TempForm.Designer.Controls. _
      Add("Forms.CommandButton.1")
   With NewCommandButton1
      .Caption = "Отмена"
      .Cancel = True
```

```
        .Height = 18
        .Width = 44
        .Left = MaxWidth + 12
        .Top = 6
    End With

'   Добавление кнопки OK
    Set NewCommandButton2 = TempForm.Designer.Controls. _
       Add("Forms.CommandButton.1")
    With NewCommandButton2
        .Caption = "OK"
        .Default = True
        .Height = 18
        .Width = 44
        .Left = MaxWidth + 12
        .Top = 28
    End With
'   Процедуры обработки событий для кнопок CommandButton
    Code = ""
    Code = Code & "Sub CommandButton1_Click()" & vbCrLf
    Code = Code & " GETOPTION_RET_VAL=False" & vbCrLf
    Code = Code & " Unload Me" & vbCrLf
    Code = Code & "End Sub" & vbCrLf
    Code = Code & "Sub CommandButton2_Click()" & vbCrLf
    Code = Code & " Dim ctl" & vbCrLf
    Code = Code & " GETOPTION_RET_VAL = False" & vbCrLf
    Code = Code & " For Each ctl In Me.Controls" & vbCrLf
    Code = Code & " If TypeName(ctl) = ""OptionButton""" _
       & " Then" & vbCrLf
    Code = Code & " If ctl Then GETOPTION_RET_VAL = " _
       & "ctl.Tag" & vbCrLf
    Code = Code & " End If" & vbCrLf
    Code = Code & " Next ctl" & vbCrLf
    Code = Code & " Unload Me" & vbCrLf
    Code = Code & "End Sub"
    With TempForm.CodeModule
        .InsertLines .CountOfLines + 1, Code
    End With

'   Изменение параметров диалогового окна
    With TempForm
        .Properties("Caption") = Title
        .Properties("Width") = NewCommandButton1.Left + _
           NewCommandButton1.Width + 10
        If .Properties("Width") < 160 Then
           .Properties("Width") = 160
           NewCommandButton1.Left = 106
           NewCommandButton2.Left = 106
        End If
        .Properties("Height") = TopPos + 24
    End With

'   Отображение диалогового окна
    VBA.UserForms.Add(TempForm.Name).Show
'   Удаление диалогового окна
    ThisWorkbook.VBProject.VBComponents.Remove VBComponent: _
       =TempForm
```

```
'  Передача выбранного параметра вызывающей процедуре
   GetOption = GETOPTION_RET_VAL
End Function
```

Функция GetOption работает достаточно быстро, учитывая количество действий, которое выполняется в фоновом режиме. Во многих компьютерах диалоговое окно отображается практически мгновенно. После выполнения своей функции диалоговое окно UserForm удаляется.

Использование функции GetOption

Функция GetOption принимает три аргумента.

- OpArray — строковый массив, содержащий названия элементов, которые будут отображены в диалоговом окне с помощью элементов управления OptionButton.
- Default — целое число, которое указывает на элемент управления OptionButton, выбранный по умолчанию в момент отображения диалогового окна UserForm. Если этот параметр имеет значение 0, то не выбран ни один из элементов управления OptionButton.
- Title — текст, который будет отображаться в строке заголовка созданного диалогового окна UserForm.

Принцип действия функции GetOption

Функция GetOption выполняет следующие операции.

1. Скрывает окно VBE, что позволяет избежать мерцания экрана в момент генерации диалогового окна UserForm и добавления кода процедур обработки событий.
2. Создает диалоговое окно UserForm и назначает его переменной TempForm.
3. Затем в диалоговое окно добавляются элементы управления OptionButton.

   Для этого используется массив, который передан в функцию с помощью аргумента OpArray. Свойство Tag элемента управления используется для сохранения индексного номера. Значение свойства Tag выбранного элемента управления возвращается функцией в качестве результата.
4. После этого функция добавляет два элемента управления CommandButton: кнопку OK и кнопку Отмена.
5. Для каждого из элементов управления CommandButton создается процедура обработки события.
6. Проводятся завершающие действия по настройке диалогового окна UserForm: функция изменяет расположение кнопок CommandButton и размер диалогового окна UserForm.
7. Диалоговое окно UserForm отображается на экране. Когда пользователь щелкает на кнопке OK, выполняется процедура CommandButton1_Click. Она определяет, какой из элементов управления OptionButton выделен, и присваивает соответствующее значение переменной GETOPTION_RET_VAL (эта переменная объявлена с областью действия Public).
8. Диалоговое окно UserForm закрывается и удаляется.

9. Функция возвращает значение переменной GETOPTION_RET_VAL в качестве результата.

Примечание

Заметным преимуществом динамического создания диалогового окна UserForm является размещение функции в единственном модуле, а также отсутствие необходимости в ссылке на библиотеку VBA Extensibility Library. Таким образом, этот модуль можно просто экспортировать (название модуля — modOptionsForm), а затем импортировать его в любую из рабочих книг и получить доступ к функции GetOption.

Следующая процедура демонстрирует использование функции GetOption. В этом случае диалоговое окно UserForm предоставляет пять вариантов выбора (которые содержатся в массиве Ops).

```
Sub TestGetOption()
   Dim Ops(1 To 5)
   Dim UserOption
   Ops(1) = "Север"
   Ops(2) = "Юг"
   Ops(3) = "Запад"
   Ops(4) = "Восток"
   Ops(5) = "Все регионы"
   UserOption = GetOption(Ops, 5, "Выберите регион")
   Debug.Print UserOption
   MsgBox Ops(UserOption)
End Sub
```

Переменная UserOption хранит индексный номер выбранного переключателя. Если пользователь щелкнет на кнопке Отмена (или нажмет клавишу <Escape>), то переменная UserOption будет установлена в значение False.

Обратите внимание на то, что значение свойства Accelerator установлено равным первому символу для каждого заголовка опции, поэтому пользователь может использовать комбинацию клавиш <Alt+*буква*> для выбора требуемого варианта. Я не пытался избежать дублирования комбинаций клавиш, поэтому пользователю придется нажать комбинацию клавиш несколько раз для выполнения выделения.

На рис. 28.10 показано диалоговое окно UserForm, которое генерируется этой функцией.

Примечание

Диалоговое окно UserForm изменяет свой размер для того, чтобы подстроиться под заданное количество элементов управления, добавленных в диалоговое окно. Теоретически функция GetOption может принимать массив любого размера. Однако на практике количество элементов массива ограничивается, поскольку размер диалогового окна не может превышать размер экрана. На рис. 28.11 показано, каким образом выглядит окно, содержащее большое количество опций.

Код обработчика событий GetOption

Ниже приводится код процедур обработчиков событий для двух элементов управления CommandButton. Этот код генерируется функцией GetOption и находится в модуле кода (для временного диалогового окна UserForm).

```
Sub CommandButton1_Click()
   GETOPTION_RET_VAL = False
   Unload Me
End Sub
Sub CommandButton2_Click()
   Dim ctl
   GETOPTION_RET_VAL = False
   For Each ctl In Me.Controls
      If TypeName(ctl) = "OptionButton" Then
         If ctl Then GETOPTION_RET_VAL = ctl.Tag
      End If
   Next ctl
   Unload Me
End Sub
```

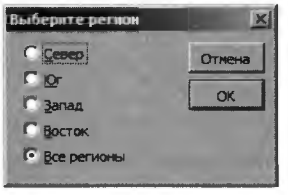


Рис. 28.10. Функция `GetOption` генерирует пользовательское окно `UserForm`

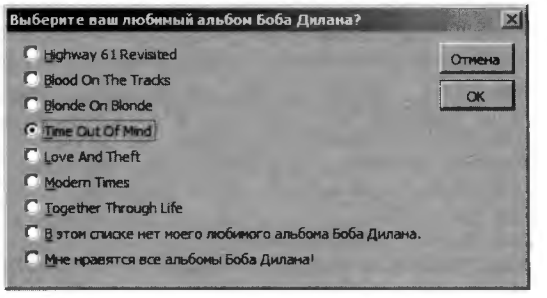


Рис. 28.11. Высота и ширина диалогового окна `UserForm` изменяются в целях компенсации количества опций и длины текста

Примечание

Поскольку диалоговое окно `UserForm` после использования удаляется, его невозможно увидеть в окне VBE. Если вы все же хотите его увидеть, преобразуйте следующую инструкцию в комментарий путем добавления перед ней символа апострофа (`'`).

```
ThisWorkbook.VBProject.VBComponents.Remove _
   VBComponent:=TempForm
```

Глава 29

Модули классов

В этой главе...

- Определение модуля класса
- Пример создания модуля класса
- Дополнительные сведения о модулях классов
- Модуль класса CSVFileClass

В данной главе описываются основные приемы использования модулей классов, а также приводятся примеры, которые помогут вам понять принципы управления модулями классов.

Определение модуля класса

Для большинства программистов на VBA концепция модуля класса — неизведанная территория. Модули классов поддерживаются в Excel 97 и более поздних версиях программы. Примеры, рассматриваемые в этой главе, помогут приподнять "завесу тайны" над этими объектами.

Модуль класса (class module) — это специальный тип модуля VBA, который можно добавить в проект. Проще говоря, модуль класса позволяет разработчику создать новый класс объектов. На этом этапе вы уже должны знать, что программирование Excel сводится к управлению объектами. Модуль класса позволяет создать новые объекты, а также соответствующие свойства, методы и события.

Перекрестная ссылка

Примеры из предыдущих глав (главы 15, 18, 19 и 22) демонстрируют применение модулей классов.

У вас может возникнуть вопрос "Нужно ли создавать новые объекты?" Ответ краток: нет. Однако как только вы поймете преимущества модулей классов, то можете рискнуть и попробовать. В большинстве случаев модуль класса служит достойной заменой функциям и процедурам, но можно найти для него и более серьезное применение. Иногда модуль класса является единственным средством получения необходимого результата.

Ниже приводится список типичных вариантов использования модулей классов.

- **Обработка событий, связанных с внедренными диаграммами.** (Соответствующий пример представлен в главе 18.)
- **Контроль над событиями уровня приложения.** В число таких событий входит активизация рабочей книги. (Соответствующие примеры находятся в главах 19 и 22.)
- **Перенастройка функций Windows API для упрощения их использования в исходном коде.** Например, можно создать класс, который упрощает определение или изменение состояния клавиш <NumLock> и <CapsLock>. Также можно создать класс, который упрощает доступ к системному реестру.
- **Выполнение одной процедуры несколькими объектами диалогового окна `UserForm`.** Обычно объект имеет собственную процедуру обработки события. Пример в главе 15 демонстрирует использование модуля класса, предназначенного для предоставления нескольким элементам управления `CommandButton` одной процедуры обработки события `Click`.
- **Создание компонентов, рассчитанных на повторное использование; их можно импортировать в другие проекты.** Как только будет создан модуль класса общего назначения, его можно импортировать в другие проекты для сокращения объема работ по разработке приложений.

Пример создания модуля класса

В данном разделе приводится пошаговая инструкция создания полезного, хотя и достаточно простого, модуля класса. Этот модуль класса создает класс `NumLock`, который имеет одно свойство (`Value`) и один метод (`Toggle`).

Определение или установка значения клавиши <NumLock> требует использования нескольких функций Windows API. Такой модуль класса предназначен для упрощения операции управления клавишей <NumLock>. Вызовы API-функций, а также необходимый код расположены в модуле класса (а не в обычном модуле кода VBA). В чем преимущества такого подхода? Работать с подобным кодом намного проще, а модуль класса можно использовать в других проектах.

Создав класс в коде VBA, определите текущее состояние клавиши <NumLock> с помощью инструкции, отображающей значение свойства `Value`.

```
MsgBox NumLock.Value
```

Также в коде можно изменить состояние клавиши <NumLock>. Например, следующий оператор включает клавишу <NumLock>:

```
NumLock.Value = True
```

Также клавишу <NumLock> можно включить с помощью метода `Toggle`.

```
NumLock.Toggle
```

Важно понимать, что модуль класса содержит код, который *определяет* объект, включая его свойства и методы. В результате предоставляется возможность создания экземпляров этого класса в модуле кода VBA общего назначения, а также использования его свойств и методов.

Для того чтобы лучше понять процесс создания модуля класса, воспользуйтесь инструкциями следующего раздела. Начнем с создания новой рабочей книги.

Вставка модуля класса

Запустите редактор VBE и выберите команду Insert⇨Class Module (Вставка⇨ дуль класса). Это приведет к добавлению пустого модуля класса, который называ Class1. Если окно Properties (Свойства) не отображено на экране, нажмите клав <F4>. После отображения окна Properties измените имя модуля класса на NumLo Class (рис. 29.1).

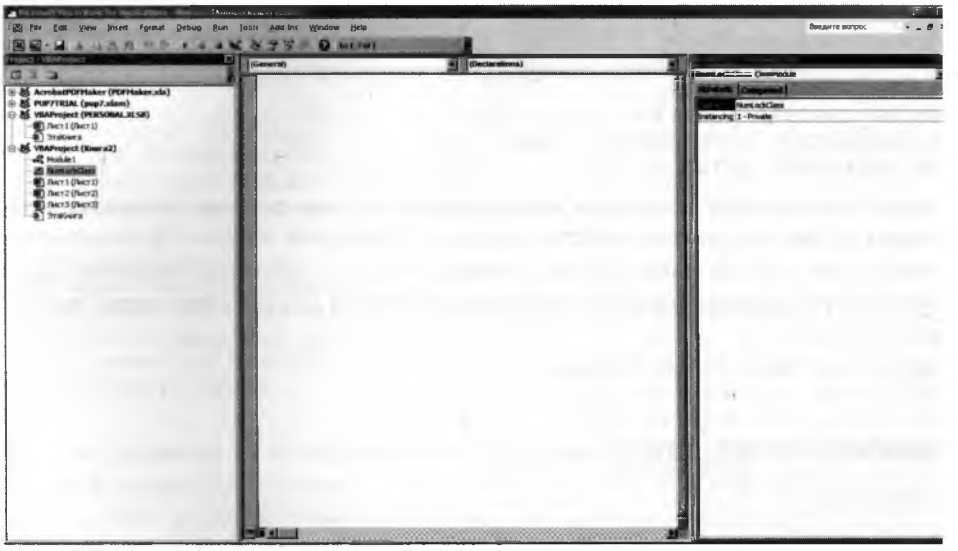


Рис. 29.1. Пустой модуль класса под именем NumLockClass

Добавление кода VBA

Далее необходимо создать код, обеспечивающий работоспособность свойства Va Чтобы определить или изменить состояние клавиши <NumLock>, модуль класса дол включать объявления функций Windows API, которые используются для установки получения состояния клавиши <NumLock>. Этот код приводится ниже.

Примечание

Используемый в этом примере код получен на основе кода, находяще на веб-сайте Microsoft. Приведенный ниже код работает только в Excel 2 Версия кода, находящаяся на компакт-диске, совместима с предыду версиями Excel.

```
'  Определение типа
Private Type OSVERSIONINFO
   dwOSVersionInfoSize As Long
   dwMajorVersion As Long
   dwMinorVersion As Long
   dwBuildNumber As Long
   dwPlatformId As Long
   szCSDVersion As String * 128
End Type
'  Определения API-функций
Private Declare PtrSafe Function GetVersionEx Lib "Kernel32" _
```

```
   Alias "GetVersionExA" _
   (lpVersionInformation As OSVERSIONINFO) As Long
Private Declare PtrSafe Sub keybd_event Lib "user32" _
   (ByVal bVk As Byte, _
   ByVal bScan As Byte, _
   ByVal dwflags As Long, ByVal dwExtraInfo As Long)
Private Declare PtrSafe Function GetKeyboardState Lib "user32" _
   (pbKeyState As Byte) As Long
Private Declare PtrSafe Function SetKeyboardState Lib "user32" _
   (lppbKeyState As Byte) As Long
' Определения констант
Const VK_NUMLOCK = &H90
Const VK_SCROLL = &H91
Const VK_CAPITAL = &H14
Const KEYEVENTF_EXTENDEDKEY = &H1
Const KEYEVENTF_KEYUP = &H2
```

После этого создайте процедуру, которая будет получать текущее состояние клавиши <NumLock>. Она определяет свойство `Value` создаваемого объекта. В качестве имени свойства можно использовать любое название. `Value` — неплохой вариант. Для того чтобы получить текущее состояние клавиши, вставьте в модуль следующую процедуру `Property Get`.

```
Property Get Value() As Boolean
'  Получение текущего состояния
   Dim keys(0 To 255) As Byte
   GetKeyboardState keys(0)
   Value = keys(VK_NUMLOCK)
End Property
```

Перекрестная ссылка

Подробно код процедур группы `Property` рассматривается далее. Для получения необходимых сведений обратитесь к разделу "Программирование свойств объектов".

Процедура, использующая функцию Windows API `GetKeyboardState`, определяет текущее состояние клавиши <NumLock>. Эта процедура вызывается каждый раз, когда код VBA запрашивает значение свойства `Value`. Например, оператор VBA, который представлен ниже, приводит к выполнению процедуры `Property Get`.

```
MsgBox NumLock.Value
```

Теперь необходимо создать процедуру, которая будет устанавливать состояние клавиши <NumLock>. Клавиша может иметь одно из двух состояний: включена или выключена. Такая задача выполняется с помощью следующей процедуры `Property Let`.

```
Property Let Value(boolVal As Boolean)
   Dim o As OSVERSIONINFO
   Dim keys(0 To 255) As Byte
   o.dwOSVersionInfoSize = Len(o)
   GetVersionEx o
   GetKeyboardState keys(0)
'  Находится ли в этом состоянии?
   If boolVal = True And keys(VK_NUMLOCK) = 1 Then Exit Property
   If boolVal = False And keys(VK_NUMLOCK) = 0 Then Exit Property
'  Переключение.
'  Имитация нажатия клавиши
   keybd_event VK_NUMLOCK, &H45, KEYEVENTF_EXTENDEDKEY Or 0, 0
```

```
'  Имитация отпускания клавиши
   keybd_event VK_NUMLOCK, &H45, KEYEVENTF_EXTENDEDKEY Or _
      KEYEVENTF_KEYUP, 0
End Property
```

Процедура `Property Let` принимает один аргумент, который может иметь либо значение `True`, либо `False`. Оператор VBA, который приводится ниже, устанавливает значение `Value` объекта `NumLock` в значение `True`. Для этого вызывается процедура `Property Let`:

```
NumLock.Value = True
```

Наконец, вам потребуется процедура переключения режима `NumLock`.

```
Sub Toggle()
'  Переключение состояния
   Dim o As OSVERSIONINFO
   o.dwOSVersionInfoSize = Len(o)
   GetVersionEx o
   Dim keys(0 To 255) As Byte
   GetKeyboardState keys(0)
'  Имитация нажатия клавиши
   keybd_event VK_NUMLOCK, &H45, KEYEVENTF_EXTENDEDKEY Or 0, 0
'  Имитация отпускания клавиши
   keybd_event VK_NUMLOCK, &H45, KEYEVENTF_EXTENDEDKEY Or _
      KEYEVENTF_KEYUP, 0
End Sub
```

Обратите внимание на то, что в качестве процедуры `Toggle` применяется стандартная процедура (а не процедуры `Property Let` либо `Property Get`). Приведенный ниже оператор VBA переключает состояние объекта `NumLock` путем вызова процедуры `Toggle`:

```
NumLock.Toggle
```

Использование модуля класса `NumLockClass`

Перед применением модуля класса `NumLockClass` необходимо создать экземпляр класса. Оператор, который находится в модуле кода VBA общего назначения (не в модуле класса), создает такой экземпляр:

```
Dim NumLock As New NumLockClass
```

Обратите внимание на то, что тип объекта указан как `NumLockClass` (т.е. используется название модуля класса). Сама переменная может иметь любое название, но `NumLock` — наиболее приемлемое.

Представленная далее процедура устанавливает свойство `Value` объекта `NumLock` в значение `True`. Это приводит к включению клавиши <NumLock>.

```
Sub NumLockOn()
   Dim NumLock As New NumLockClass
   NumLock.Value = True
End Sub
```

Следующая процедура отображает окно сообщения, которое указывает на текущее состояние клавиши <NumLock> (`True` — если клавиша включена и `False` — в противном случае).

```
Sub GetNumLockState()
   Dim NumLock As New NumLockClass
```

```
   MsgBox NumLock.Value
End Sub
```

Представленная ниже процедура переключает клавишу <NumLock>.

```
Sub ToggleNumLock()
   Dim NumLock As New NumLockClass
   NumLock.Toggle
End Sub
```

Наконец, последняя процедура переключает состояние клавиши <NumLock> без использования метода `Toggle`.

```
Sub ToggleNumLock2()
   Dim NumLock As New NumLockClass
   NumLock.Value = Not NumLock.Value
End Sub
```

Понятно, что использование класса `NumLock` намного проще, чем обращение к API-функциям. Также ранее созданный модуль класса можно использовать в любом другом проекте.

Компакт-диск

Полностью завершенный модуль класса для этого примера находится на прилагаемом к книге компакт-диске в файле `keyboard classes.xlsm`. Здесь же находятся модули классов для обнаружения и установки состояния клавиш <Caps Lock> и <Scroll Lock>.

Дополнительные сведения о модулях классов

Пример предыдущего раздела продемонстрировал способы создания нового объекта, обладающего свойством `Value` и методом `Toggle`. Но объект может иметь любое количество свойств, методов и событий.

Имя, которое используется для модуля класса, является одновременно и именем объекта класса. По умолчанию модули классов получают такие имена, как `Class1`, `Class2` и т.д. Модулю класса требуется более точное имя.

Программирование свойств объектов

Многие объекты обладают как минимум одним свойством. Однако объект может иметь любое количество свойств. После определения свойства его используют в коде, для чего применяется стандартный синтаксис с разделителем-точкой:

`объект.свойство`

Параметр редактора VBE Auto List Members (Автоматическая вставка объектов) применяется и к объектам, определенным в модуле класса. Это упрощает определение свойств и методов при написании кода.

Свойства объекта, определяемого в модуле класса, могут предназначаться только для чтения, только для записи, а также для чтения и записи. Свойство, предназначенное только для чтения, определяется с помощью процедуры `Property Get`. Ниже приведен пример подобной процедуры.

```
Property Get FileNameOnly() As String
   FileNameOnly = ""
   For i = Len(FullName) To 1 Step -1
      Char = Mid(FullName, i, 1)
```

```
      If Char = "\" Then
         Exit Function
      Else
         FileNameOnly = Char & FileNameOnly
      End If
   Next i
End Property
```

Несложно заметить, что процедура `Property Get` работает как функция. Код вычисляет и возвращает значение свойства, которое соответствует названию процедуры. В приведенном примере в качестве имени процедуры используется `FileNameOnly`. Процедура возвращает имя файла, которое содержится в значении общедоступной переменной `FullName`. Например, если переменная `FullName` содержит значение `c:\data\myfile.txt`, то процедура возвращает значение `myfile.txt`. Процедура `FileNameOnly` вызывается тогда, когда код VBA обращается к свойству объекта.

Для свойств, ориентированных на чтение и запись, необходимо создать две процедуры: `Property Get` (отвечает за чтение значения свойства) и `Property Let` (устанавливает значение свойства). Значение, определенное свойству, рассматривается в качестве последнего (или единственного) аргумента процедуры `Property Get`.

Ниже приводятся два примера подобных процедур.

```
Dim XLFile As Boolean
Property Get SaveAsExcelFile() As Boolean
   SaveAsExcelFile = XLFile
End Property
Property Let SaveAsExcelFile(boolVal As Boolean)
   XLFile = boolVal
End Property
```

Примечание

Если свойство имеет объектный тип данных, используется `Property Set` вместо `Property Let`.

Переменная, объявленная в модуле класса с областью действия `Public`, может также применяться в качестве свойства объекта. В представленном выше примере избавимся от процедур `Property Get` и `Property Let` и заменим их на объявление переменной на уровне модуля.

```
Public SaveAsExcelFile As Boolean
```

Если необходимо активизировать свойство, предназначенное только для записи, воспользуйтесь процедурой `Property Let` без соответствующей процедуры `Property Get`.

Предыдущие примеры предполагают существование булевой переменной `XLFile`, которая определена на уровне модуля. Процедура `Property Get` возвращает значение этой переменной в качестве значения свойства. Если объект называется `FileSys`, то следующий оператор будет отображать текущее значение свойства `SaveAsExcelFile`:

```
MsgBox FileSys.SaveAsExcelFile
```

С другой стороны, процедура `Property Let` принимает аргумент и использует значение аргумента для изменения значения свойства. Например, можно создать оператор (приведенный ниже), который будет устанавливать свойство `SaveAsExcelFile` в значение `True`.

```
FileSys.SaveAsExcelFile = True
```

В данном случае значение `True` передается процедуре `Property Let`, которая и изменяет значение свойства.

В предыдущих примерах используется переменная уровня модуля — `XLFile`. Именно эта переменная используется для хранения значения свойства. Такую переменную необходимо создавать для каждого свойства, которое объявляется в модуле класса.

Примечание

Процедуры свойств подчиняются всем правилам именования процедур. Может оказаться, что VBA не позволит использовать в названиях процедур отдельные зарезервированные слова. Если при создании процедуры свойства отображается сообщение о синтаксической ошибке, рекомендуется изменить ее название.

Программирование методов объектов

Метод объекта класса программируется подобно другим процедурам. Для этого используются ключевые слова `Sub` или `Function`, которые размещаются в модуле класса. Как правило, объект имеет методы, однако может существовать и без них. Для вызова метода в коде используются стандартные соглашения:

```
объект.метод
```

Как и любой другой метод VBA, метод объекта класса выполняет определенные действия. Следующая процедура представляет метод, который сохраняет рабочую книгу в одном из двух форматов. Используемый формат определяется значением свойства `XLFile`. Нетрудно заметить, что в этой процедуре нет ничего необычного.

```
Sub SaveFile()
   If XLFile Then
   ActiveWorkbook.SaveAs FileName:=FName, _
      FileFormat:=xlWorkbookNormal
   Else
      ActiveWorkbook.SaveAs FileName:=FName, _
         FileFormat:=xlCSV
   End If
End Sub
```

Класс `CSVFileClass`, рассматриваемый в следующем разделе, существенно прояснит концепцию свойств и методов объекта, определенного в модуле класса.

События модуля класса

Каждый модуль класса поддерживает два события: `Initialize` и `Terminate`. Событие `Initialize` возникает при создании нового экземпляра класса, а событие `Terminate` — при уничтожении объекта. Событие `Initialize` можно использовать для установки значений свойств, принятых по умолчанию.

Шаблон для процедур обработки событий выглядит следующим образом.

```
Private Sub Class_Initialize()
'  Здесь находится код инициализации
End Sub
Private Sub Class_Terminate()
'  Здесь находится код уничтожения
End Sub
```

Объект *уничтожается* (и освобождается выделенная для него память), когда процедура или модуль, в котором он объявлен, завершает свое выполнение. Объект можно уничтожить в любой момент, присвоив ему значение Nothing. Следующий оператор уничтожает объект, который называется MyObject:

```
Set MyObject = Nothing
```

Модуль класса CSVFileClass

Пример, приведенный в этом разделе, определяет такой класс объектов, как CSVFileClass. Описанный в этом модуле класс имеет два свойства и два метода.

- **Свойства**
  - ExportRange — предназначено для чтения и записи. Указывает диапазон ячеек рабочего листа, который будет экспортирован в файл формата CSV.
  - ImportRange — предназначено для чтения и записи. Указывает диапазон, в который будет импортирована информация из файла формата CSV.
- **Методы**
  - Import — используется для импортирования информации из файла в формате CSV, который определяется значением аргумента CSVFileName. Информация импортируется в диапазон, обозначенный свойством ImportRange.
  - Export — используется для экспортирования диапазона, определенного значением свойства ExportRange, в файл формата CSV, который указывается значением аргумента CSVFileName.

Компакт-диск

Рассматриваемый в этом разделе пример находится на прилагаемом к книге компакт-диске в файле csv class.xlsm.

Переменные уровня модуля класса

Модуль класса должен поддерживать собственные локальные переменные, которые отражают значения свойств объекта. Модуль класса CSVFileClass содержит две переменные, которые используются для хранения значений свойств. Такие переменные объявляются в начале модуля.

```
Private RangeToExport As Range
Private ImportToCell As Range
```

Здесь RangeToExport — это объект Range, который представляет экспортируемый диапазон. ImportToCell является объектом Range, представляющим верхнюю левую ячейку того диапазона, в который будут импортироваться данные из файла в формате CSV. Эти переменные получают значения с помощью процедур Property Get и Property Let (см. следующий раздел).

Процедуры свойств

Процедуры свойств модуля класса CSVFileClass показаны ниже. Процедуры Property Get используются для получения значения переменной, а процедуры Property Let — для установки значения переменной.

```
Property Get ExportRange() As Range
   Set ExportRange = RangeToExport
End Property
Property Let ExportRange(rng As Range)
   Set RangeToExport = rng
End Property
Property Get ImportRange() As Range
   Set ImportRange = ImportToCell
End Property
Property Let ImportRange(rng As Range)
   Set ImportToCell = rng
End Property
```

Процедуры методов

Модуль класса CSVFileClass содержит две процедуры, которые представляют методы класса. Эти процедуры рассматриваются в следующем разделе.

Процедура Export

Процедура Export вызывается каждый раз при обращении к методу Export. Процедура принимает один аргумент: полное имя файла, который будет хранить экспортируемый диапазон. Процедура обеспечивает базовый уровень обработки ошибок. Например, проверяется наличие значения у свойства ExportRange. Для этого проверяется значение переменной RangeToExport. Процедура создает обработчик для возникающих ошибок.

```
Sub Export(CSVFileName)
'  Экспортирует диапазон в файл CSV
   If RangeToExport Is Nothing Then
      MsgBox "Экспортируемый диапазон не определен"
      Exit Sub
   End If

   On Error GoTo ErrHandle
   Application.ScreenUpdating = False
   Set ExpBook = Workbooks.Add(xlWorksheet)
   RangeToExport.Copy
   Application.DisplayAlerts = False

   With ExpBook
      .Sheets(1).Paste
      .SaveAs FileName:=CSVFileName, FileFormat:=xlCSV
      .Close SaveChanges:=False
   End With
   Application.CutCopyMode = False
   Application.ScreenUpdating = True
   Application.DisplayAlerts = True
   Exit Sub
ErrHandle:
   ExpBook.Close SaveChanges:=False
```

```
    Application.CutCopyMode = False
    Application.ScreenUpdating = True
    Application.DisplayAlerts = True
    MsgBox "Ошибка " & Err & vbCrLf & vbCrLf & Error(Err), _
       vbCritical, "Ошибка метода экспортирования"
End Sub
```

Процедура `Export` при выполнении копирует указанный с помощью переменной `RangeToExport` диапазон во временную рабочую книгу. После этого рабочая книга сохраняется в файле формата CSV, и файл закрывается. Так как обновление экрана не выполняется, пользователь не видит происходящего. Если возникнет ошибка (например, будет указано неверное имя файла), процедура перейдет к метке `ErrHandle` и отобразит окно сообщения, которое содержит номер ошибки и ее описание.

Процедура `Import`

Процедура `Import` импортирует данные из файла CSV, который указан с помощью аргумента `CSVFileName`. Содержимое файла копируется в диапазон, указанный с помощью значения переменной `ImportToCell`. Эта переменная используется для хранения значения свойства `ImportRange`. В данном случае обновление экрана не выполняется — пользователь не должен иметь возможности наблюдать за работой процедуры. Как и `Export`, процедура `Import` содержит базовые функции обработки ошибок.

```
Sub Import(CSVFileName)
'  Импортирование файла CSV в диапазон
    If ImportToCell Is Nothing Then
       MsgBox "Диапазон для импорта не определен "
       Exit Sub
    End If

    If CSVFileName = "" Then
       MsgBox "Не определено имя импортируемого файла"
       Exit Sub
    End If

    On Error GoTo ErrHandle
    Application.ScreenUpdating = False
    Application.DisplayAlerts = False
    Workbooks.Open CSVFileName
    Set CSVFile = ActiveWorkbook
    ActiveSheet.UsedRange.Copy Destination:=ImportToCell
    CSVFile.Close SaveChanges:=False
    Application.ScreenUpdating = True
    Application.DisplayAlerts = True
    Exit Sub
ErrHandle:
    CSVFile.Close SaveChanges:=False
    Application.ScreenUpdating = True
    Application.DisplayAlerts = True
    MsgBox "Ошибка " & Err & vbCrLf & vbCrLf & Error(Err), _
       vbCritical, "Ошибка метода импортирования "
End Sub
```

Использование класса `CSVFileClass`

Для того чтобы создать экземпляр класса `CSVFileClass` в собственном коде VBA, необходимо создать переменную, которая имеет тип `CSVFileClass`. Ниже приведен пример объявления такой переменной.

```
Dim CSVFile As New CSVFileClass
```

Иногда может потребоваться объявить переменную указанного типа, а также создать объект в тот момент, когда он будет запрашиваться в коде. Для этого используются операторы `Dim` и `Set`.

```
Dim CSVFile As CSVFileClass
'  Здесь может находиться другой код
Set CSVFile = New CSVFileClass
```

Преимуществом применения операторов `Dim` и `Set` является то, что объект не создается до тех пор, пока в коде не встретится соответствующий оператор `Set`. При этом очевидна экономия памяти: если объект не нужен, память под него не выделяется. Например, код может следовать алгоритму, который точно определяет, когда необходимо создавать объект. Кроме того, использование оператора `Set` позволяет создавать несколько экземпляров одного класса.

После создания экземпляра объекта допускается вводить другие операторы, которые будут обращаться к свойствам и методам созданного объекта, определенного в модуле класса.

Параметр Auto List Members (рис. 29.2) может применяться по отношению ко всем объектам. После ввода имени переменной с точкой в конце будет отображен список свойств и методов, которые определены для этого объекта.

Рис. 29.2. Параметр Auto List Members отображает перечень доступных свойств и методов

Следующая процедура демонстрирует сохранение текущего выделенного диапазона ячеек в файл CSV, который называется `temp.csv`. Этот файл находится в той же папке, что и рабочая книга.

```
Sub ExportARange()
   Dim CSVFile As New CSVFileClass
   With CSVFile
      .ExportRange = ActiveWindow.RangeSelection
      .Export CSVFileName:=ThisWorkbook.Path & "\temp.csv"
   End With
End Sub
```

Использовать структуру `With-End With` не обязательно. Например, процедура может выглядеть следующим образом.

```
Sub ExportARange()
   Dim CSVFile As New CSVFileClass
   CSVFile.ExportRange = ActiveWindow.RangeSelection
   CSVFile.Export CSVFileName:=ThisWorkbook.Path & "\temp.csv"
End Sub
```

Представленная далее процедура демонстрирует операцию импортирования файла CSV. Данные размещаются начиная с активной ячейки.

```
Sub ImportAFile()
   Dim CSVFile As New CSVFileClass
   With CSVFile
   On Error Resume Next
      .ImportRange = ActiveCell
      .Import CSVFileName:=ThisWorkbook.Path & "\temp.csv"
   End With
   If Err <> 0 Then _
      MsgBox "Невозможно импортировать " & _
         ThisWorkbook.Path & "\temp.csv"
End Sub
```

VBA может работать с большим количеством экземпляров класса. Следующий код создает массив из трех классов `CSVFileClass`.

```
Sub Export3Files()
   Dim CSVFile(1 To 3) As New CSVFileClass
   CSVFile(1).ExportRange = Range("A1:A20")
   CSVFile(2).ExportRange = Range("B1:B20")
   CSVFile(3).ExportRange = Range("C1:C20")
   For i = 1 To 3
      CSVFile(i).Export CSVFileName:="Файл" & i & ".csv"
   Next i
End Sub
```

Глава 30

Работа с цветом

В этой главе...

- Определение цвета
- Оттенки серого
- Экспериментирование с цветами
- Темы документа
- Работа с фигурами
- Изменение цветов диаграммы

В главе рассматриваются вопросы работы с цветом в Excel 2010.

Определение цвета

Работа с цветом в Excel 2010 не относится к категории тривиальных задач. Более того, она достаточно сложна. Зачастую написанный вами макрос, который должен изменять цвет ячейки или объекта, не оправдывает возложенных на него надежд. Поэтому, чтобы избежать разочарований в дальнейшем, прочтите эту главу.

Одним из наиболее существенных изменений в Excel 2007 явился отказ от ограниченной палитры, состоящей из 56 цветов. Этот ограниченный набор цветов применялся для раскрашивания фона ячеек, оформления текста и диаграмм. Конечно, можно было изменить эти цвета, но не существовало способа расширить набор из 56 цветов, применяемых для оформления рабочей книги.

Но все течет и меняется. С появлением Excel 2007 количество цветов, применяемых для оформления рабочей книги, стало практически неограниченным. Конечно, предел существует, но он настолько велик (16 777 216 цветов), что можно считать количество цветов неограниченным.

В VBA цвет задается с помощью десятичного значения цвета, которое может изменяться от `0` до `16777215`. Ниже приводится оператор VBA, с помощью которого для фона выбирается темный красно-коричневый цвет.

```
ActiveCell.Interior.Color = 5911168
```

Также для чаще всего применяемых цветов в VBA имеются предопределенные константы. Например, значение константы vbRed равно 255 (десятичное значение, соответствующее красному цвету), а константа vbGreen имеет значение 65280.

Чтобы определить около 17 миллионов цветов, потребуется соответствующее количество констант, запомнить которые не в состоянии никто. Поэтому для определения цвета используют процентные соотношения красного, зеленого и синего цветов — цветовую модель RGB.

Цветовая модель RGB

Цветовая система RGB представляет собой комбинацию различных по интенсивности красного, зеленого и синего цветов. Значение каждого из этих цветов изменяется в диапазоне от 0 до 255. Общее количество возможных цветов подсчитывается по формуле 256×256×256 = 16 777 216. Если значения всех цветовых компонентов равны 0, получаем чисто черный цвет. Если же величины всех компонентов равны 255, на выходе имеем абсолютно белый цвет. Если же величины компонентов равны 128 (середина диапазона), получаем нейтральный серый цвет. Оставшиеся 16 777 213 комбинаций представляют все остальные цвета.

Для определения цвета с помощью цветовой модели RGB применяется функция VBA под названием RGB. Эта функция принимает три аргумента, которые представляют красный, зеленый и синий компоненты цвета. В результате ее выполнения возвращается десятичное значение цвета.

В приведенном ниже операторе используется функция RGB для определения цвета, который использовался в предыдущем разделе (темный красно-коричневый, значение 5911168).

```
ActiveCell.Interior.Color = RGB(128, 50, 90)
```

В табл. 30.1 приведены значения RGB и десятичные значения цвета для некоторых часто применяемых цветов.

Таблица 30.1. Примеры определения цвета

Имя	Красный компонент	Зеленый компонент	Синий компонент	Значение цвета
Black (Черный)	0	0	0	0
White (Белый)	255	255	255	16777215
Red (Красный)	255	0	0	255
Green (Зеленый)	0	255	0	65280
Blue (Синий)	0	0	255	16711680
Yellow (Желтый)	255	255	0	65535
Pink (Розовый)	255	0	255	16711935
Turquoise (Бирюзовый)	0	255	255	16776960
Brown (Коричневый)	153	51	0	13209
Indigo (Темно-синий)	51	51	153	10040115
80% Gray (80% серый)	51	51	51	3355443

Цветовая модель HSL

Если при выборе цвета в Excel щелкнуть на ссылке Другие цвета (More Colors), на экране появится диалоговое окно Цвета (Colors). Воспользовавшись вкладкой Спектр (Custom), пользователь может выбрать одну из двух цветовых моделей, применяемых для определения цвета: RGB и HSL. На рис. 30.1 показано диалоговое окно Цвета с выбранной цветовой моделью HSL.

Рис. 30.1. Выбор цвета в цветовой модели HSL

В цветовой модели HSL выбор цвета осуществляется с помощью трех параметров: Оттенок (Hue), Насыщение (Saturation) и Яркость (Luminance). Как и в случае с цветами RGB, каждый из этих параметров изменяется в диапазоне от 0 до 255. Каждому цвету RGB соответствует определенный цвет HSL, а каждому цвету HSL отвечает эквивалентное десятичное значение цвета. Другими словами, каждый из 16 777 216 цветов может быть определен с помощью какой-либо из трех цветовых моделей: RGB, HSL или десятичной.

Несмотря на то что в диалоговом окне Цвета (Colors) можно определить цвет с помощью цветовой модели HSL, помните о том, что при этом задается *лишь* область, в которой Excel поддерживает цветовую модель HSL. А при определении цвета с помощью VBA используется десятичное значение цвета. Для получения подобного значения можно использовать функцию `RGB` (при работе с цветовой моделью RGB). А вот функции `HSL` в VBA не предусмотрено.

Преобразование цветов

Если известны значения трех компонентов цвета, преобразование цвета RGB в десятичное значение цвета не составит особого труда. Просто воспользуйтесь функцией VBA под названием `RGB`. Предположим, что имеются три переменные (`r`, `g` и `b`), каждая из которых представляет значение компонента цвета, изменяющееся от 0 до 255. Для вычисления эквивалентного десятичного значения цвета воспользуйтесь следующим оператором:

```
DecimalColor = RGB(r, g, b)
```

Для выполнения подобного преобразования с помощью формулы рабочего листа создайте следующую функцию-оболочку VBA:

```
Function RGB2DECIMAL(R, G, B)
'  Преобразование из RGB в десятичное значение цвета
   RGB2DECIMAL = RGB(R, G, B)
End Function
```

В следующем примере функции рабочего листа предполагается, что три значения цвета находятся в ячейках A1:C1.

```
=RGB2DECIMAL(A1,B1,C1)
```

Процедура преобразования десятичного значения цвета в красный, зеленый и синий компоненты цвета более сложная. Ниже приводится пример функции, которая возвращает значения этих компонентов в виде трехэлементного массива.

```
Function DECIMAL2RGB(ColorVal) As Variant
'  Преобразует значение цвета в триплет RGB
'  Возвращает 3-элементный массив типа Variant
   DECIMAL2RGB = Array(ColorVal \ 256 ^ 0 And 255,
      ColorVal \ 256 ^ 1 And 255, ColorVal \ 256 ^ 2 And 255)
End Function
```

Для использования функции DECIMAL2RGB в составе формулы рабочего листа следует вводить эту формулу в виде формулы массива, состоящего из трех ячеек. Например, предположим, что ячейка A1 содержит десятичное значение цвета. Для преобразования цветового значения в эквивалентные компоненты RGB выберите горизонтальный диапазон, состоящий из трех ячеек, а затем введите указанную ниже формулу. Нажмите комбинацию клавиш <Ctrl+Shift+Enter> для создания формулы массива, не используя при этом фигурных скобок.

```
{=DECIMAL2RGB(A1)}
```

Если используется вертикальный массив, состоящий из трех ячеек, следует транспонировать массив, как показано ниже.

```
{=TRANSPOSE(DECIMAL2RGB(A1))}
```

На рис. 30.2 показан результат выполнения функции DECIMAL2RGB.

Компакт-диск

На прилагаемом компакт-диске в файле `color conversion functions.xlsm` находится рабочая книга, включающая следующие функции преобразования цвета: DECIMAL2RGB, DECIMAL2HSL, HSL2RGB, RGB2DECIMAL, RGB2HSL и HSL2DECIMAL.

Десятичные значения цвета

У многих пользователей возникает вопрос о том, каким образом распределяются 16 777 216 десятичных значений цвета. Значение `0` соответствует черному цвету, значение `16777216` — белому цвету, а что можно сказать о промежуточных цветах?

Чтобы получить представление обо всех десятичных значениях цвета, воспользуйтесь указанным ниже кодом, который построен на основе двух вложенных циклов `For-Next`.

```
Sub GenerateColorValues()
   Dim Red As Long, Blue As Long, Green As Long
   Dim AllColors(0 To 16777215) As Long
   Dim ColorNum As Long
```

```
    ColorNum = 0
    For Blue = 0 To 255
       For Green = 0 To 255For Red = 0 To 255
          For Red = 0 To 255For Red = 0 To 255
             AllColors(ColorNum) = RGB(Red, Blue, Green)
             ColorNum = ColorNum + 1
          Next Red
       Next Green
    Next Blue
End Sub
```

В результате выполнения этой процедуры происходит заполнение массива `AllColors` десятичными значениями для всех цветов, используемых Excel.

	A	B	C	D	E	F	G	H	I	J
1	Десятичное число		Десятичное-в-RGB				Десятичное-в-HSL			
2	Значение цвета		R	G	B		H	S	L	
3	0		0	0	0		0	0	0	
4	167 772		92	143	2		58	248	72	
5	335 544		184	30	5		6	242	94	
6	503 316		20	174	7		82	235	90	
7	671 088		112	61	10		21	213	61	
8	838 860		204	204	12		42	227	108	
9	1 006 632		40	92	15		71	184	54	
10	1 174 404		132	235	17		63	221	126	
11	1 342 176		224	122	20		21	213	122	
12	1 509 948		60	10	23		244	182	35	
13	1 677 720		152	153	25		43	183	89	
14	1 845 492		244	40	28		2	231	136	
15	2 013 264		80	184	30		71	184	107	
16	2 181 036		172	71	33		12	173	102	
17	2 348 808		8	215	35		91	237	112	
18	2 516 580		100	102	38		44	117	70	
19	2 684 352		192	245	40		53	232	142	
20	2 852 124		28	133	43		91	166	80	
21	3 019 896		120	20	46		244	182	70	
22	3 187 668		212	163	48		30	167	130	
23	3 355 440		48	51	51		128	8	50	

Рис. 30.2. Пример использования функций рабочего листа `DECIMAL2RGB` и `DECIMAL2HSL`

Оттенки серого

В процессе создания рабочих листов и диаграмм следует помнить о том, что далеко не у каждого пользователя имеется цветной принтер. И даже если ваша диаграмма будет распечатана на цветном принтере, вполне возможно, что в дальнейшем она будет ксерокопирована, отправлена по факсу либо попадет на стол дальтоника (известно, что этим недостатком страдают 8% мужского населения).

В процессе печати цветных изображений на черно-белых принтерах происходит преобразование цвета в оттенки серого. При определенном везении преобразование выполняется удачно, и цвета будут неплохо смотреться, будучи преобразованными в оттенки серого. Но так бывает далеко не всегда. Например, зачастую после преобразования цветной гистограммы в "серую" ее столбцы сливаются.

Цвета образуются путем смешивания красного, зеленого и синего компонентов. Черному цвету соответствует результат выполнения функции `RGB(0, 0, 0)`, белому — `RGB(255, 255, 255)`, нейтрально-серому — `RGB(128, 128, 128)`. Благодаря использованию этой функции генерируется 256 оттенков серого цвета.

Для создания в диапазоне ячеек шкалы серого цвета, состоящей из 256 оттенков, используется указанная ниже процедура. В результате ее выполнения в различные оттенки серого цвета (от черного до белого) окрашивается фон ячеек, находящихся в диапазоне от A1 до A256. Для просмотра всего диапазона можно уменьшить масштаб.

```
Sub GenerateGrayScale()
   Dim r As Long
   For r = 0 To 255
      Cells(r + 1, 1).Interior.Color = RGB(r, r, r)
   Next r
End Sub
```

На рис. 30.3 показан результат заливки увеличенных ячеек градациями серого цвета.

Преобразование цветов в оттенки серого

Преобразовать цвет в оттенки серого можно путем усреднения значений красного, зеленого и синего компонентов цвета, причем полученному серому цвету будет соответствовать единственное значение. При этом не учитывается то, что различные цвета воспринимаются человеческим глазом по-разному. Например, зеленый цвет кажется ярче красного, а красный — ярче синего.

В процессе экспериментов по восприятию цвета был выработан следующий "рецепт", предписывающий преобразование цветовых значений RGB в оттенки серого:

- 28,7% красного компонента;
- 58,9% зеленого компонента;
- 11,4% синего компонента.

В качестве примера рассмотрим цветовое значение `16751001` (оттенок фиолетового цвета), которое соответствует значению `RGB(153, 153, 255)`. Учитывая рассмотренные ранее соображения, получим следующие значения RGB:

- красный: 28,7%.153 = 44;
- зеленый: 58,9%.153 = 90;
- синий: 11,4%.255 = 29.

Сумма этих значений равна 163. Таким образом, значение RGB в оттенках серого, соответствующее цветовому значению 16751001, равно `RGB(163, 163, 163)`.

Ниже приведен код функции VBA, в качестве аргумента которой принимается десятичное значение цвета. В результате ее выполнения возвращается десятичное значение серого цвета.

```
Function Grayscale(color)
   Dim r As Long, g As Long, b As Long
   r = (color \ 256 ^ 0 And 255) * 0.287
   g = (color \ 256 ^ 1 And 255) * 0.589
   b = (color \ 256 ^ 2 And 255) * 0.114
   Grayscale = RGB(r + g + b, r + g + b, r + g + b)
End Function
```

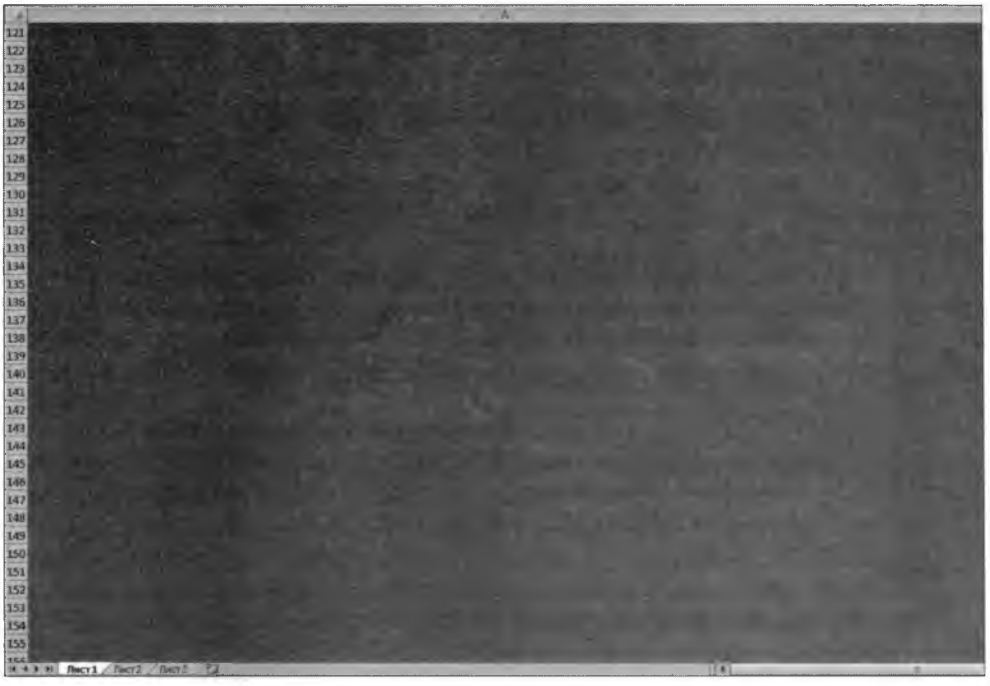


Рис. 30.3. В ячейке отображается 256 оттенков серого цвета

Просмотр диаграмм в оттенках серого

К сожалению, средство предварительного просмотра в Excel не предусматривает преобразование в градации серого. И если в вашем распоряжении имеется черно-белый лазерный принтер, вам все равно придется довольствоваться предварительным просмотром в цвете.

В подобной ситуации может помочь вкладка **Лист** (Sheet) диалогового окна **Параметры страницы** (Page Setup). Для отображения этой вкладки щелкните на значке открытия диалогового окна, находящегося в группе Параметры страницы (Page Setup) вкладки **Разметка страницы** (Page Layout). Теперь осталось установить флажок **Черно-белая** (Black And White), и все диаграммы будут печататься в истинном черно-белом цвете (не в градациях серого). Цвета преобразуются в узоры, состоящие из черно-белых оттенков. Обратите внимание: эта настройка применяется только к диаграммам и другим графическим объектам. При печати в черно-белом режиме цвета ячеек игнорируются.

Ниже представлена методика, которая позволит преобразовать цвета встроенных диаграмм в градации серого.

1. Выделите диаграмму.
2. Нажмите комбинацию клавиш <Ctrl+C> для копирования диаграммы в буфер обмена.
3. Щелкните на ячейке и выберите команду **Главная⇨Буфер обмена⇨Вставить⇨Рисунок** (Home⇨Clipboard⇨Paste⇨Picture).
4. Выделите вставленный рисунок и выберите команду **Работа с рисунками⇨Формат⇨Изменение⇨Цвет** (Picture Tools⇨Format⇨Adjust⇨Color), затем в разделе **Перекрасить** (Recolor) появившейся на экране коллекции выберите пункт **Градации серого** (Grayscale) (рис. 30.4).

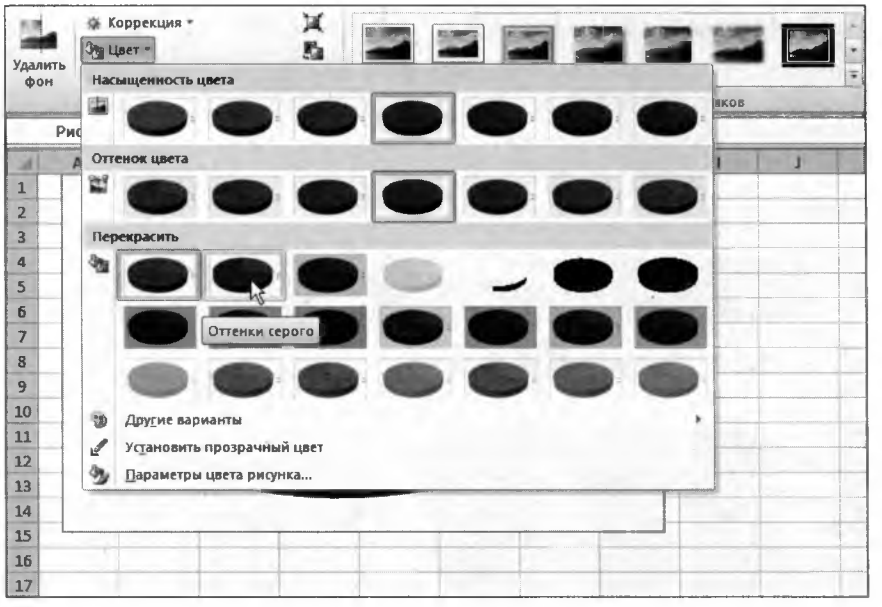


Рис. 30.4. Преобразование цветов рисунка в градации серого

Эти действия можно автоматизировать с помощью макроса, код которого приводится ниже. Процедура `ShowChartAsGrayScale` копирует активную пиктограмму в виде рисунка, а затем преобразует цвета в оттенки серого. Если полученные оттенки вас устраивают, можно удалить рисунок.

```
Sub ShowChartAsGrayScale()
'   Копирование активной диаграммы в рисунок в оттенках серого;
'   только для внедренных диаграмм
    If ActiveChart Is Nothing Then
       MsgBox "Выбор диаграммы."
       Exit Sub
    End If
    ActiveChart.Parent.CopyPicture
    ActiveChart.Parent.TopLeftCell.Select
    ActiveSheet.Pictures.Paste
    ActiveSheet.Pictures(ActiveSheet.Pictures.Count). _
    ShapeRange.PictureFormat.ColorType = msoPictureGrayscale
End Sub
```

Компакт-диск

Рабочая книга, содержащая рассматриваемый в этом разделе пример, находится на прилагаемом к книге компакт-диске в файле `chart to grayscale picture.xlsm`.

Совет

Не забывайте о встроенных стилях диаграмм в градациях серого. Эти стили автоматизированы для наилучшего отображения отдельных элементов диаграмм.

Экспериментирование с цветами

На рис. 30.5 показана рабочая книга, созданная автором в процессе экспериментирования с цветами. Если вы не совсем понимаете, каким образом функционирует цветовая модель RGB, потратьте немного времени на эту рабочую книгу, после чего все покажется не столь уж и сложным.

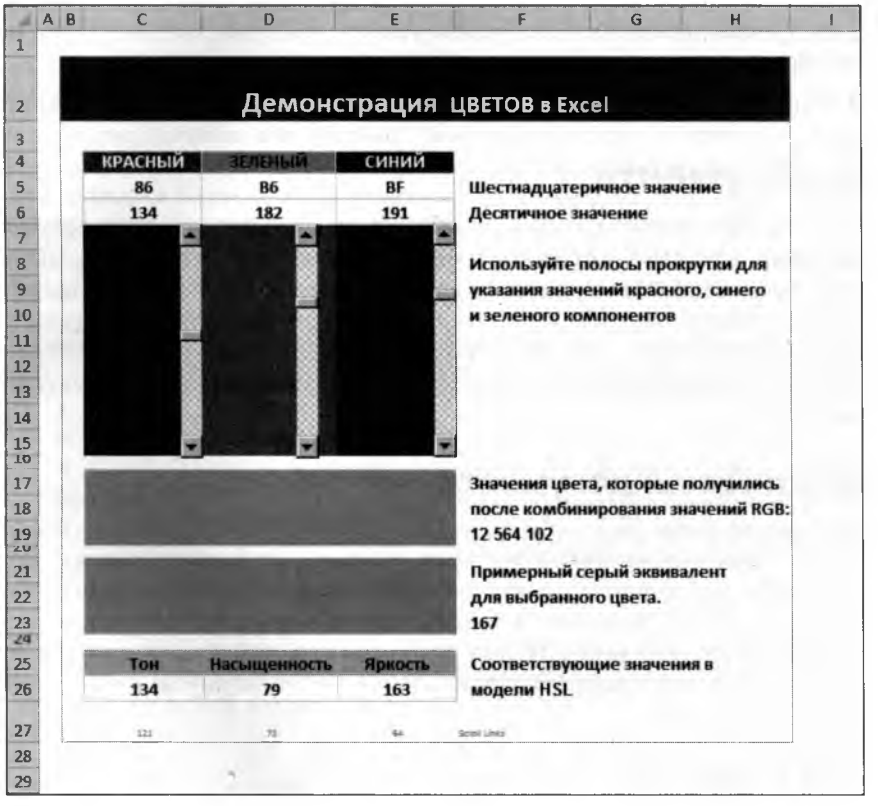


Рис. 30.5. Эта рабочая книга позволит просматривать различные сочетания красного, зеленого и синего цветов

Компакт-диск

Рассматриваемая в этом разделе рабочая книга находится на прилагаемом к книге компакт-диске в файле `RGB color demo.xlsm`.

В окне этой рабочей книги находятся вертикальные полосы прокрутки, каждая из которых управляет фоновым цветом диапазона. Используйте эти полосы для определения значений красного, зеленого и синего компонентов цвета в диапазоне от 0 до 255. С помощью полос прокрутки можно изменить ряд областей рабочего листа.

- В ячейках, находящихся над полосами прокрутки, отображаются значения компонентов цвета в шестнадцатеричном (00–FF) и десятичном формате (0–255). Шестнадцатеричные значения цвета RGB часто используются для определения цветов в HTML-документах.

- В диапазонах ячеек, находящихся сбоку от полос прокрутки, можно изменить интенсивность окрашивания (в зависимости от положения ползунка полосы прокрутки). Это значение соответствует числовой величине цветового компонента.
- В диапазоне ячеек, находящемся ниже полос прокрутки, указывается комбинированный цвет, определяемый с помощью указанных пользователем значений RGB.
- В ячейке отображается десятичное значение цвета.
- В следующем диапазоне ячеек можно увидеть, как будет выглядеть цвет после его преобразования в оттенки серого.
- В диапазоне ячеек также отображаются соответствующие значения цвета HSL.

Темы документа

Еще в Excel 2007 появилось такое замечательное свойство, как тема документа. Теперь достаточно единственного щелчка мышью для изменения внешнего облика всего документа. Тема документа включает следующие три компонента: цвета, шрифты и эффекты (для графических объектов). Благодаря темам документа пользователи могут создавать презентабельные, оформленные в едином стиле, документы. Обратите внимание на то, что тема применяется ко всей рабочей книге, а не только к активному рабочему листу.

Концепция темы документа

В комплект поставки Microsoft Office 2010 входит 40 тем документа. Также можно загрузить или создать дополнительные темы. На ленте находится несколько коллекций стилей. (В качестве примера может служить коллекция **Стили диаграмм** (Chart Styles).) Набор стилей, входящих в состав каждой коллекции, зависит от темы, выбранной для данного документа. И если для документа выбирается другая тема, изменяется его внешний вид в результате применения цветов, шрифтов и эффектов, входящих в состав новой темы.

Компакт-диск

Если вы не хотите самостоятельно просматривать темы документа, на прилагаемом компакт-диске откройте рабочую книгу `document theme demo.xlsx`. Здесь вы найдете диапазон ячеек, отображающий цвет каждой темы, две фигуры, текст (с применением обычного шрифта и шрифта заголовка), а также диаграмму. Выберите команду Разметка страницы⇨Темы⇨Темы (Page Layout⇨Themes⇨Themes Gallery) для просмотра изменений рабочего листа после применения каждой выбранной темы.

Можно также смешивать и сопоставлять элементы темы. Например, можно воспользоваться цветами из одной темы, шрифтами из другой темы и эффектами из третьей темы. Также можно создать новый набор цветов или шрифтов. Пользовательские темы могут быть сохранены и применены в дальнейшем к другим рабочим книгам.

Примечание

Концепция тем документа основана на теории, согласно которой пользователи склонны выполнять в документах незначительное форматирование, которое не охватывается темами. Если пользователь применил свои цвета и шрифты, которые не входят в состав текущей темы, результаты подобного форматирования не изменяются после выбора новой темы документа. Поэтому весьма вероятно создание небрежно оформленного документа, включающего несочетаемые цвета и слишком много разных шрифтов.

Цвета темы документа

После применения цвета к ячейке или объекту можно его изменить, воспользовавшись раскрывающейся коллекцией **Цвета темы**, которая появляется после щелчка на значке **Цвет заливки** в группе **Шрифт** вкладки **Главная** (рис. 30.6). В окне этой коллекции отображается 60 цветов темы (10 столбцов и 6 строк), а также 10 дополнительных стандартных цветов. После щелчка на ссылке **Другие цвета** (More Colors) появится диалоговое окно **Цвета** (Color), в котором можно выбрать любой из 16777216 доступных цветов.

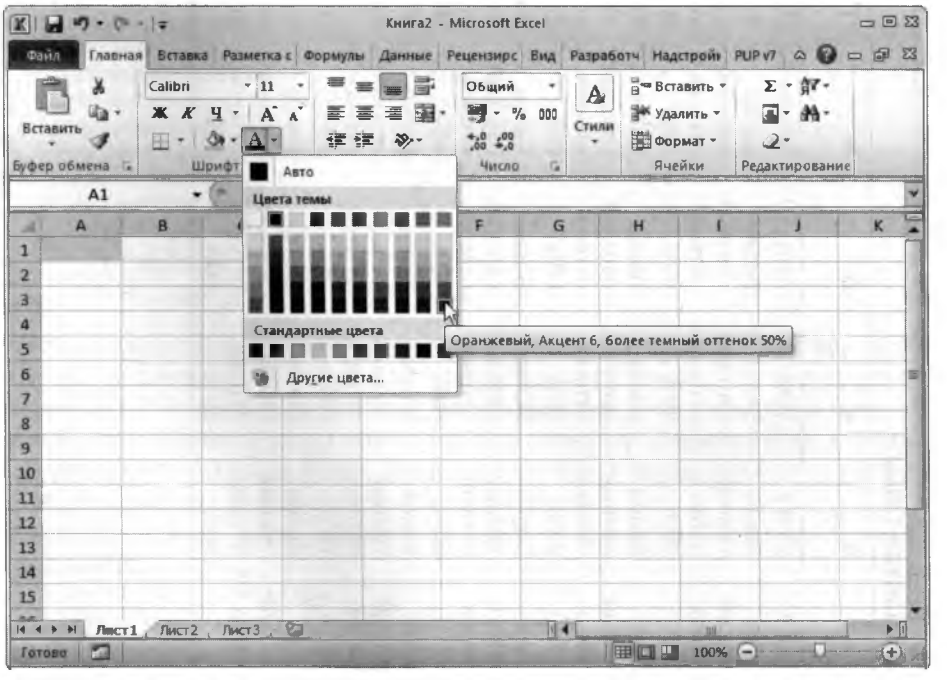


Рис. 30.6. В этом окне выбираются цвета темы

С помощью всплывающих подсказок идентифицируется 60 цветов темы. Например, цвет, находящийся во второй строке шестого столбца, называется "Акцент 2, более светлый оттенок 20%". В табл. 30.2 приведены названия всех 60 цветов тем. Одни цвета темы называются более светлыми оттенками (Tint), а другие — более темными оттенками (Shade).

Таблица 30.2. Названия цветов темы

Строка/ столбец	1	2	3	4	5	6	7	8	9	10
1	Фон 1	Текст 1	Фон 2	Текст 2	Акцент 1	Акцент 2	Акцент 3	Акцент 4	Акцент 5	Акцент 6
2	Более темный оттенок 5%	Более светлый оттенок 50%	Более темный оттенок 10%	Более светлый оттенок 80%	Более светлый оттенок 80%	Более светлый оттенок 80%	Более светлый оттенок 80%	Более светлый оттенок 80%	Более светлый оттенок 80%	Более светлый оттенок 80%
3	Более темный оттенок 15%	Более светлый оттенок 35%	Более темный оттенок 25%	Более светлый оттенок 80%	Более светлый оттенок 60%	Более светлый оттенок 60%	Более светлый оттенок 60%	Более светлый оттенок 60%	Более светлый оттенок 60%	Более светлый оттенок 60%
4	Более темный оттенок 25%	Более светлый оттенок 25%	Более темный оттенок 50%	Более светлый оттенок 80%	Более светлый оттенок 40%	Более светлый оттенок 40%	Более светлый оттенок 40%	Более светлый оттенок 40%	Более светлый оттенок 40%	Более светлый оттенок 40%
5	Более темный оттенок 35%	Более светлый оттенок 15%	Более темный оттенок 75%	Более темный оттенок 25%	Более темный оттенок 25%	Более темный оттенок 25%	Более темный оттенок 25%	Более темный оттенок 25%	Более темный оттенок 25%	Более темный оттенок 25%
6	Более темный оттенок 50%	Более светлый оттенок 5%	Более темный оттенок 90%	Более темный оттенок 50%	Более темный оттенок 50%	Более темный оттенок 50%	Более темный оттенок 50%	Более темный оттенок 50%	Более темный оттенок 50%	Более темный оттенок 50%

Обратите внимание: названия цветов остаются одинаковыми, даже если выбирается другая тема документа. Десять цветов темы документа отображаются в верхней строке (четыре цвета "текст/фон" и шесть цветов "акцент"). Для каждого из этих цветов применяется пять вариаций более светлых и более темных оттенков.

Примечание

Если выбрать команду Разметка страницы⇨Темы⇨Цвета⇨Создать новые цвета темы (Page Layout⇨Themes⇨Colors⇨Create New Theme Colors), на экране появится два дополнительных цвета темы: Гиперссылка (Hyperlink) и Просмотренная гиперссылка (Followed Hyperlink). Эти цвета используются в случае создания гиперссылки и не отображаются в окне выбора цвета темы.

После прочтения этого раздела возникает желание создать макрос, который позволяет изменить цвет заливки и шрифта для диапазона ячеек. Ниже приводится подобный макрос, который был записан после выбора диапазона ячеек. В качестве цвета заливки был выбран "Акцент 2, более темный оттенок 25%", а для цвета шрифта — "Текст 2, более светлый оттенок 80%".

```
Sub ChangeColors()
   With Selection.Interior
      .Pattern = xlSolid
      .PatternColorIndex = xlAutomatic
      .ThemeColor = xlThemeColorAccent2
      .TintAndShade = -0.249977111117893
      .PatternTintAndShade = 0
   End With
   With Selection.Font
      .ThemeColor = xlThemeColorLight2
      .TintAndShade = 0.799981688894314
   End With
End Sub
```

Я решил отказаться от применения трех свойств, имеющих отношение к узорам (`Pattern`, `PatternColorIndex` и `PatternTintAndShade`). Они имеют отношение к "старомодным" узорам, применяемым для оформления ячеек (эти узоры по-прежнему поддерживаются). Данные узоры можно выбрать на вкладке Заливка (Fill) в диалоговом окне Формат ячеек (Format Cells). Указанные свойства поддерживают существующие узоры, которые могут входить в состав диапазона ячеек.

Записанный макрос (после удаления связанных с узорами свойств) приведен ниже.

```
Sub ChangeColors()
   With Selection.Interior
'       (Акцент 2, более темный оттенок 25%)
      .ThemeColor = xlThemeColorAccent2
      .TintAndShade = -0.249977111117893
   End With
   With Selection.Font
'       (Текст 2, более светлый оттенок 80%)
      .ThemeColor = xlThemeColorLight2
      .TintAndShade = 0.799981688894314
   End With
End Sub
```

Таким образом, для определения цвета используются свойства `ThemeColor` и `TintAndShade`. Значение свойства `ThemeColor` формируется согласно простому правилу:

номер столбца в таблице цветов темы. Значение свойства `TintAndShade` формируется по более сложному правилу. Рассмотрим это правило подробнее.

Значение свойства `TintAndShade` может изменяться от –1 до +1. Значение –1 соответствует черному цвету, а значение +1 — белому цвету. Если свойству `TintAndShade` присвоено значение 0, получаем *цвет без оттенков* (чистый цвет) Другими словами, отрицательные значения свойства `TintAndShade` соответствуют более темным оттенкам цвета вплоть до черного. Положительные значения свойства `TintAndShade` соответствуют более светлым оттенкам цвета вплоть до белого. Для получения значения свойства `TintAndShade`, которое соответствует определенному оттенку цвета темы, обратитесь к табл. 30.2.

Если для характеристики цвета применяется термин "Более светлый оттенок" (Tint), значение свойства `TintAndShade` будет положительным. Если для характеристики цвета применяется термин "Более темный оттенок" (Shade), значение свойства `TintAndShade` будет отрицательным.

Примечание

Я не знаю, почему при записи макроса значения свойства `TintAndShade` выражаются множеством знаков после запятой. Как по мне, то между значениями свойства `TintAndShade`, равными `-0.249977111117893` и `-0.25`, особой разницы нет.

Компакт-диск

Для демонстрации способов изменения цвета с помощью свойства `TintAndShade` откройте рабочую книгу `tintandshade demo.xlsm`, находящуюся на прилагаемом компакт-диске (рис. 30.7). Укажите начальный цвет, после чего макрос отобразит этот цвет с применением 50 уровней значений свойства `TintAndShade`, которые варьируются от -1 до +1. Также отображается десятичное значение цвета и красный, зеленый и синий компоненты цвета (в виде диаграммы).

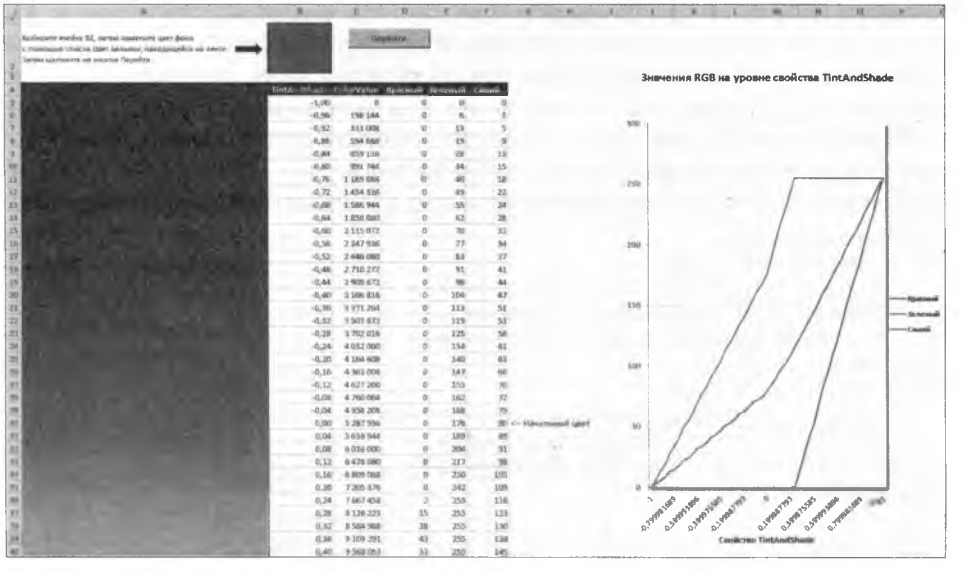


Рис. 30.7. Эта рабочая книга демонстрирует, каким образом свойство `TintAndShade` влияет на цвет

Отображение всех цветов темы

Я написал макрос, который в выбранном диапазоне ячеек отображает все 60 вариаций цветов темы.

```
Sub ShowThemeColors()
   Dim r As Long, c As Long
   For r = 1 To 6
      For c = 1 To 10
         With Cells(r, c).Interior
         .ThemeColor = c
         Select Case c
            Case 1 ' Текст/фон 1
            Select Case r
               Case 1: .TintAndShade = 0
               Case 2: .TintAndShade = -0.05
               Case 3: .TintAndShade = -0.15
               Case 4: .TintAndShade = -0.25
               Case 5: .TintAndShade = -0.35
               Case 6: .TintAndShade = -0.5
            End Select
         Case 2 ' Текст/фон 2
            Select Case r
               Case 1: .TintAndShade = 0
               Case 2: .TintAndShade = 0.5
               Case 3: .TintAndShade = 0.35
               Case 4: .TintAndShade = 0.25
               Case 5: .TintAndShade = 0.15
               Case 6: .TintAndShade = 0.05
            End Select
         Case 3 ' Текст/фон 3
            Select Case r
               Case 1: .TintAndShade = 0
               Case 2: .TintAndShade = -0.1
               Case 3: .TintAndShade = -0.25
               Case 4: .TintAndShade = -0.5
               Case 5: .TintAndShade = -0.75
               Case 6: .TintAndShade = -0.9
            End Select
         Case Else ' Текст/фон 4 и Акцент 1-6
            Select Case r
               Case 1: .TintAndShade = 0
               Case 2: .TintAndShade = 0.8
               Case 3: .TintAndShade = 0.6
               Case 4: .TintAndShade = 0.4
               Case 5: .TintAndShade = -0.25
               Case 6: .TintAndShade = -0.5
            End Select
         End Select
         Cells(r, c) = .TintAndShade
         End With
      Next c
   Next r
End Sub
```

На рис. 30.8 показан результат выполнения процедуры `ShowThemeColors` (этот результат лучше выглядит в цвете). Если выбрать другую тему документа, цвета будут изменены в соответствии с новой темой.

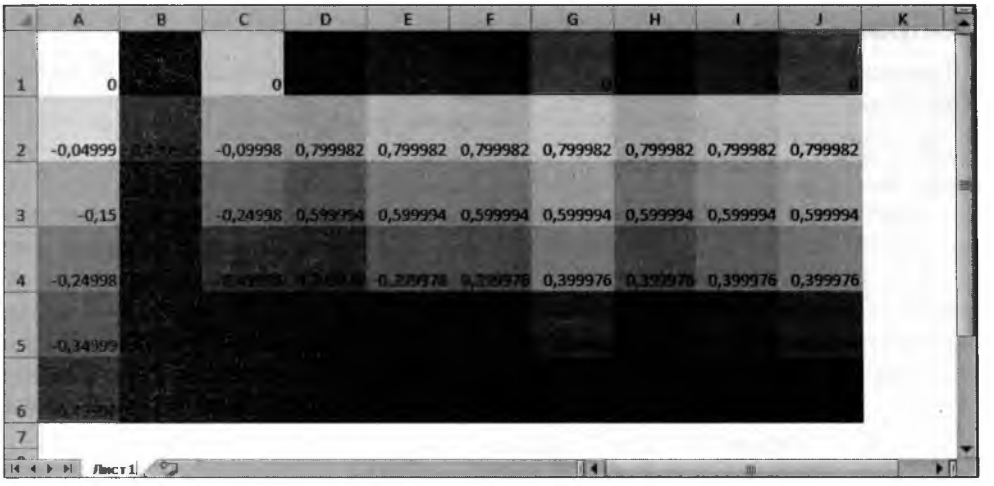


Рис. 30.8. Цвета темы были получены с помощью макроса VBA

Компакт-диск

Рассматриваемый в этом разделе пример находится на прилагаемом к книге компакт-диске в файле `generate theme colors.xlsm`.

До сих пор рассматривалось изменение цвета заливки диапазона путем установки значения свойства `Color` объекта `Interior`. Как уже отмечалось, использование функции VBA `RGB` упрощает выполнение этой задачи. Приведенные ниже два оператора демонстрируют способ изменения цвета заливки диапазона (оба оператора дают один и тот же результат).

```
Range("A1:F24").Interior.Color = 5913728
Range("A1:F24").Interior.Color = RGB(128, 60, 90)
```

Важно понимать, что присваивание цвета подобным образом не приводит к тому, что он станет цветом темы. Другими словами, если пользователь переходит к новой теме документа, в диапазоне A1:F24 цвета не изменяются. Для изменения цветов ячеек таким образом, чтобы они были совместимы с темами, воспользуйтесь свойствами `ThemeColor` и (необязательно) `TintAndShade`.

Работа с фигурами

До сих пор рассматривалось изменение цвета диапазона ячеек. В этом разделе вы найдете примеры изменения цвета объектов `Shape`, определяющих фигуры. Для добавления фигуры на рабочий лист в Excel используется команда **Вставка⇨Иллюстрации⇨Фигуры** (Insert⇨Illustrations⇨Shapes).

На рис. 30.9 показан объект `Shape`, который был включен в рабочий лист. По умолчанию этот объект называется `Стрелка вправо 1` (Right Arrow 1). Номер в имени фигуры изменяется (в зависимости от количества фигур, включенных на рабочий лист). Например, если ранее на рабочий лист были включены две фигуры (произвольного стиля), для новой фигуры используется имя `Стрелка вправо 3`.

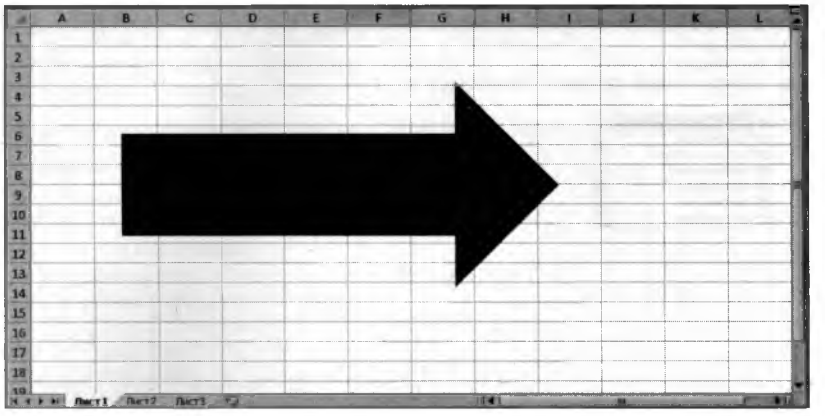


Рис. 30.9. Объект Shape на рабочем листе

Фоновый цвет фигуры

Фоновый цвет объекта Shape определяется с помощью свойства RGB. Для получения десятичного значения цвета, используемого для заливки фона фигуры, применяется следующий оператор:

```
MsgBox ActiveSheet.Shapes("Стрелка вправо 1").Fill.ForeColor.RGB
```

На первый взгляд этот оператор кажется достаточно сложным, но мы рассмотрим его по частям. Свойство Fill объекта Shape возвращает объект FillFormat. Свойство ForeColor объекта FillFormat возвращает объект ColorFormat. Таким образом, свойство RGB фактически воздействует на объект ColorFormat, и именно оно содержит десятичное значение цвета.

Примечание

Для чего же в этом примере используется свойство ForeColor? Многие разработчики кода VBA, включая автора этой книги, для изменения фонового цвета объекта используют свойство BackColor. Если объект залит с помощью узора, свойство BackColor применяется для определения второго цвета. Если же объект Shape не заливается с помощью узора, свойство ForeColor используется для управления цветом фона.

При работе с объектами Shape создаваемый пользователем код обычно выполняет несколько действий. В этом случае целесообразно использовать объектные переменные. В приведенном ниже примере кода создается объектная переменная под названием Shp.

```
Dim Shp As Shape
Set Shp = ActiveSheet.Shapes("Стрелка вправо 1")
MsgBox Shp.Fill.ForeColor.RGB
```

Совет

Использование объектной переменной обеспечивает еще одно преимущество — возможность обращения к параметру VBE Auto List Members (Автоматическая вставка объектов). Благодаря этому свойству на экране отображаются доступные свойства и объекты по мере ввода кода (рис. 30.10). Это особенно полезно при работе с объектами Shape, поскольку выполняемые при этом действия не фиксируются функцией записи макросов Excel.

Рис. 30.10. Ввод оператора с помощью свойства Auto List Members

Если вы планируете работать исключительно с цветами фигуры, создайте переменную для объекта фигуры `ColorFormat`, как показано ниже.

```
Dim ShpCF As ColorFormat
Set ShpCF = ActiveSheet.Shapes("Стрелка вправо 1").Fill.ForeColor
MsgBox ShpCF.RGB
```

Свойство `RGB` объекта `ColorFormat` контролирует цвет фигуры. Ниже описаны дополнительные свойства этого объекта. Если вы незнакомы с цветами тем документов, обратитесь к разделу "Понятие о цветах темы документа".

- `ObjectThemeColor`. Число в диапазоне от 1 до 10, которое представляет цвет темы (цвет в первом ряду сетки цвета темы размером 10×6).
- `SchemeColor`. Число, находящееся в диапазоне от 0 до 80, которое представляет цвет в виде индекса в текущей цветовой схеме. В данном случае речь идет о цветах из старой 56-цветной палитры, поэтому целесообразность применения этого свойства весьма сомнительна.
- `TintAndShade`. Число в диапазоне от –1 до +1, представляющее оттенки цвета темы.
- `Type`. Число, представляющее тип объекта `ColorFormat`. Как уже отмечалось, это свойство, предназначенное для чтения, всегда равно 1 и представляет цветовую систему RGB.

Изменение фонового цвета фигуры не оказывает влияние на цвет контура фигуры. Для изменения цвета контура следует воспользоваться объектом `ColorFormat`, относящимся к объекту фигуры `LineFormat`. Приведенный ниже код окрашивает в красный цвет фоновый цвет и контур объекта `Shape`.

```
Dim Shp As Shape
Set Shp = ActiveSheet.Shapes("Стрелка вправо 1")
Shp.Fill.ForeColor.RGB = RGB(255, 0, 0)
Shp.Line.ForeColor.RGB = RGB(255, 0, 0)
```

Ниже приводится альтернативный код, который позволяет достичь такого же эффекта путем использования объектных переменных.

```
Dim Shp As Shape
Dim FillCF As ColorFormat
Dim LineCF As ColorFormat
Set Shp = ActiveSheet.Shapes("Стрелка вправо 1")
Set FillCF = Shp.Fill.ForeColor
Set LineCF = Shp.Line.ForeColor
FillCF.RGB = RGB(255, 0, 0)
LineCF.RGB = RGB(255, 0, 0)
```

Имейте в виду, что указанный выше код не приводит к созданию цветов, совместимых с темами документа. Для указания цветов, совместимых с темами, воспользуйтесь свойством `SchemeColor` и (необязательно) свойством `TintAndShade`.

Фигуры и цвета темы

Для применения цветов темы к фигуре воспользуйтесь свойствами `ObjectThemeColor` и `TintAndShade` объекта фигуры `Forecolor`. Следующий код устанавливает цвет фигуры "Акцент 4, более светлый оттенок 40%."

```
With ActiveSheet.Shapes(1).Fill.ForeColor
   .ObjectThemeColor = msoThemeColorAccent4
   .TintAndShade = 0.4
End With
```

В результате применения этого кода получается цвет, который отличается от цвета, генерируемого с помощью кнопки **Заливка цветом** (Fill Color), находящейся на ленте.

К сожалению, реализация тем документов в программах Microsoft далека от совершенства. Например, цвета темы диапазона не соответствуют цветам темы фигуры. На рис. 30.11 показан диапазон B2:D8, для которого выбран цвет заливки "Акцент 2, более светлый оттенок 80%." На рабочем листе также находится треугольная фигура, которая залита цветом, заданным по умолчанию.

Поставленная задача звучит достаточно просто. В частности, цвет заливки фигуры следует сделать равным цвету заливки диапазона. Для этого воспользуемся следующим кодом.

```
Sub ColorShape()
   With ActiveSheet.Shapes(1).Fill.ForeColor
      .ObjectThemeColor = Range("B2:D8").Interior.ThemeColor
      .TintAndShade = Range("B2:D8").Interior.TintAndShade
   End With
End Sub
```

На рис. 30.12 показан результат выполнения процедуры `ColorShape`. Хотя цвет фигуры подобен цвету диапазона, на самом деле они не являются идентичными. Если же применить цвет темы "Акцент 2, более светлый оттенок 80%" к фигуре путем щелчка на кнопке ленты **Цвет заливки** (Fill Color), цвет фигуры будет совпадать с цветом диапазона.

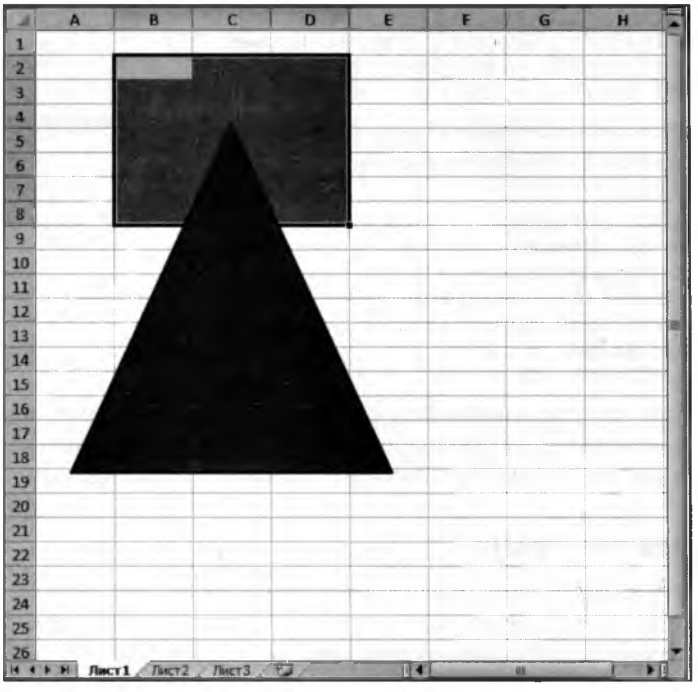


Рис. 30.11. А теперь напишем код, который зальет треугольник цветом диапазона

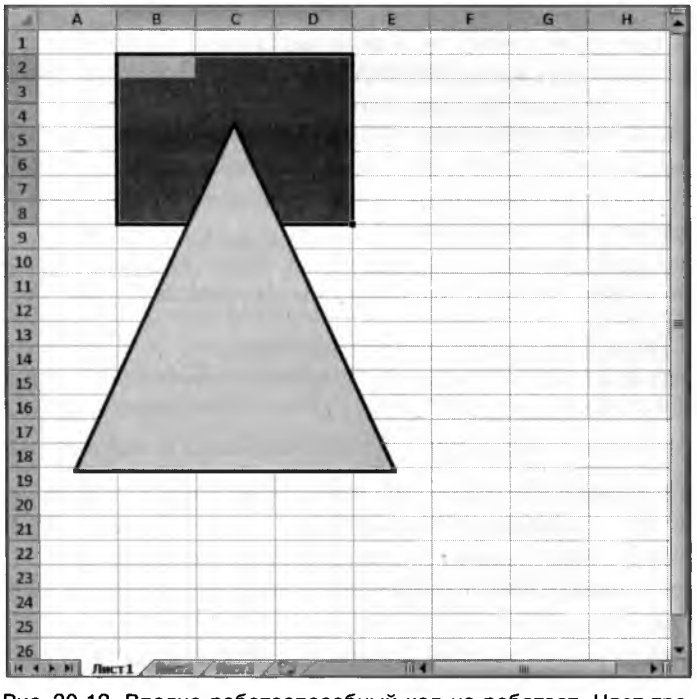


Рис. 30.12. Вполне работоспособный код не работает. Цвет треугольника отличается от цвета диапазона

Как видите, используемый нами метод выравнивания цветов фигуры и диапазона оказался неработоспособным. Окрасить диапазон с помощью свойств объекта фигуры `Interior` невозможно.

Решить эту проблему можно с помощью "старой" объектной модели. Приведенный ниже оператор выравнивает цвета фигуры и диапазона. Этот цвет не будет изменяться в случае изменения темы документа.

```
ActiveSheet.Shapes(1).Fill.ForeColor.RGB = _
   Range("B1:D8").Interior.Color
```

После ряда экспериментов с цветами фигур и диапазонов я пришел к следующим выводам:

- в Excel предлагается 15 цветов темы для фигур и только 12 цветов темы для диапазонов;
- значение `ThemeColor` для диапазона обычно (но не всегда) совпадает с величиной `ObjectThemeColor` для фигуры;
- свойство `TintAndShade` объекта `FillFormat` для фигуры всегда равно 0, даже если его устанавливать с помощью кода.

Да, похоже, что цвета, используемые для окрашивания фигур (и диаграмм), отличаются от цветов, применяемых для заливки ячеек. Вряд ли это приведет к серьезным проблемам, поскольку на практике точного совпадения цветов ячеек и объектов обычно не требуется.

Другие типы заливки фигур

Для фигур могут также применяться другие виды заливок, например заливки градиентом, рисунком или текстурой. На рис. 30.13 представлены примеры различных заливок, применяемых по отношению к объекту `Shape`.

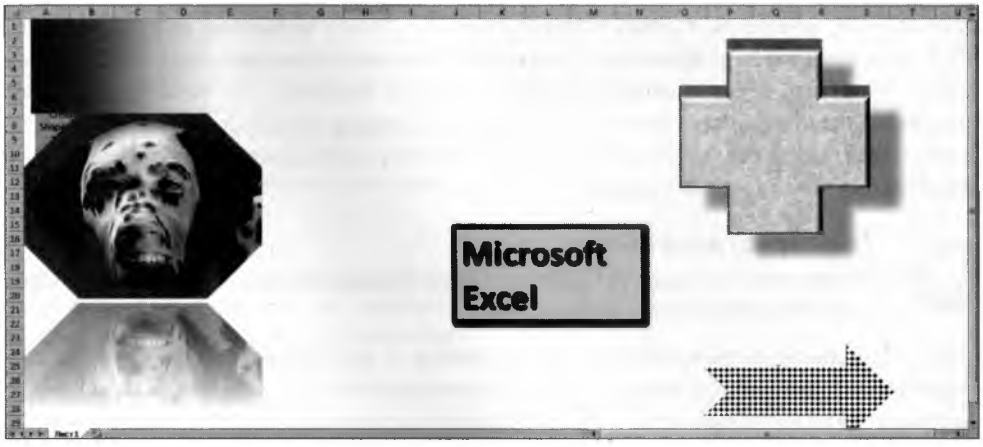


Рис. 30.13. Фигуры, сгенерированные с помощью VBA

Компакт-диск

Все примеры, рассматриваемые в этом разделе, доступны на прилагаемом к книге компакт-диске в файле `shape object colors.xlsm`.

Изменение цветов диаграммы

В этом разделе описываются способы изменения цвета диаграммы. При этом важнее всего идентифицировать отдельные элементы диаграммы, которые будут изменяться. Другими словами, следует идентифицировать объект, а затем установить его определенные свойства.

На рис. 30.14 показана простая гистограмма Диаграмма 1. Эта диаграмма состоит из двух рядов данных, легенды и заголовка диаграммы.

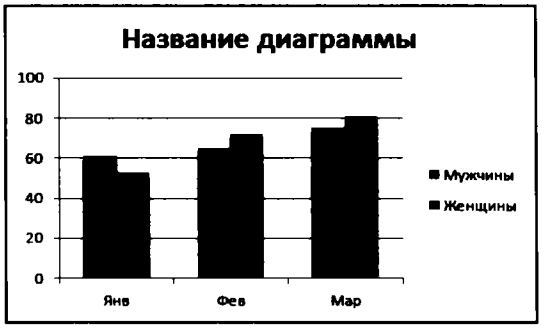


Рис. 30.14. Пример простой гистограммы

Ниже показан оператор VBA, изменяющий цвет первого ряда данных на красный.

```
ActiveSheet.ChartObjects("Диаграмма 1").Chart.
   SeriesCollection(1).Format.Fill.ForeColor.RGB = vbRed
```

У непосвященных этот оператор вызывает чувство растерянности, поскольку включает очень много объектов. Ниже рассматривается иерархия этих объектов.

Активный лист содержит коллекцию ChartObjects. В этой коллекции имеется один объект ChartObject под названием Диаграмма 1. Свойство Chart объекта ChartObject возвращает объект Chart. Объект Chart включает коллекцию SeriesCollection, и один объект Series в этой коллекции содержит индекс 1. Свойство Format объекта Series возвращает объект ChartFormat. Свойство Fill объекта ChartFormat возвращает объект FillFormat. Свойство ForeColor объекта FillFormat возвращает объект ColorFormat. Свойству RGB объекта ColorFormat присваивается значение красного цвета.

Перекрестная ссылка

Обратитесь к главе 18 для получения дополнительных сведений о работе с диаграммами посредством VBA.

Еще один способ записи предыдущего оператора — использование объектных переменных для идентификации отдельных объектов (а также выявления связей между объектами).

```
Sub ChangeSeries1Color
   Dim MyChartObject As ChartObject
   Dim MyChart As Chart
   Dim MySeries As Series
   Dim MyChartFormat As ChartFormat
   Dim MyFillFormat As FillFormat
   Dim MyColorFormat As ColorFormat
'  Создание объектов
```

```
    Set MyChartObject = ActiveSheet.ChartObjects("Диаграмма 1")
    Set MyChart = MyChartObject.Chart
    Set MySeries = MyChart.SeriesCollection(1)
    Set MyChartFormat = MySeries.Format
    Set MyFillFormat = MyChartFormat.Fill
    Set MyColorFormat = MyFillFormat.ForeColor
'   Изменение цвета
    MyColorFormat.RGB = vbRed
End Sub
```

Свойство RGB принимает десятичное значение цвета, определенного с помощью встроенной константы VBA. В следующем списке перечислены связанные с цветом свойства объекта ColorFormat.

- ObjectThemeColor. Число между 0 и 16, которое представляет цвет темы. Эти значения в VBA кодируются с помощью констант. Например, константа msoThemeColorAccent3 содержит значение 7.
- TintAndShade. Число между –1 и +1, которое представляет более светлый или темный оттенок цвета темы.

В предыдущем разделе уже упоминались проблемы, возникшие из-за различий между цветами, используемыми для заливки диапазонов и фигур. Те же проблемы возникают при работе с цветами, используемыми для заливки диаграмм.

Компакт-диск

Примеры из этого раздела находятся на прилагаемом к книге компакт-диске в файле chart colors.xlsm.

Можно также определять цветовые градиенты. Ниже приводится пример, в котором предопределенный градиент применяется ко второму ряду данных в диаграмме. Обратите внимание на то, что для настройки градиента используется объект FillFormat.

```
Sub AddPresetGradient()
    Dim MyChart As Chart
    Set MyChart = ActiveSheet.ChartObjects("Диаграмма 1").Chart
    With MyChart.SeriesCollection(1).Format.Fill
        .PresetGradient _
            Style:=msoGradientHorizontal, _
            Variant:=1, _
            PresetGradientType:=msoGradientFire
    End With
End Sub
```

Работа с другими элементами диаграммы выполняется аналогичным образом. Ниже приводится процедура, которая изменяет цвета в областях диаграммы и построения, используя цвета из текущей темы документа.

```
Sub RecolorChartAndPlotArea()
    Dim MyChart As Chart
    Set MyChart = ActiveSheet.ChartObjects("Диаграмма 1").Chart
    With MyChart
        .ChartArea.Format.Fill.ForeColor.ObjectThemeColor = _
            msoThemeColorAccent6
        .ChartArea.Format.Fill.ForeColor.TintAndShade = 0.9
        .PlotArea.Format.Fill.ForeColor.ObjectThemeColor = _
            msoThemeColorAccent6
        .PlotArea.Format.Fill.ForeColor.TintAndShade = 0.5
```

```
   End With
End Sub
```

В этом последнем примере раздела каждому элементу диаграммы присваивается случайный цвет. Применение этого макроса приведет к тому, что ваша диаграмма будет выглядеть плачевно, но зато вы ознакомитесь со способами изменения цвета отдельных элементов диаграммы. Для определения используемого цвета процедура `UseRandomColors` применяет простую функцию `RandomColor`.

```
Sub UseRandomColors()
   Dim MyChart As Chart
   Set MyChart = ActiveSheet.ChartObjects("Диаграмма 4").Chart
   With MyChart
      .ChartArea.Format.Fill.ForeColor.RGB = RandomColor
      .PlotArea.Format.Fill.ForeColor.RGB = RandomColor
      .SeriesCollection(1).Format.Fill.ForeColor.RGB = _
         RandomColor
      .SeriesCollection(2).Format.Fill.ForeColor.RGB = _
         RandomColor
      .Legend.Font.Color = RandomColor
      .ChartTitle.Font.Color = RandomColor
      .Axes(xlValue).MajorGridlines.Border.Color = RandomColor
      .Axes(xlValue).TickLabels.Font.Color = RandomColor
      .Axes(xlValue).Border.Color = RandomColor
      .Axes(xlCategory).TickLabels.Font.Color = RandomColor
      .Axes(xlCategory).Border.Color = RandomColor
   End With
End Sub
Function RandomColor()
   RandomColor = Application.RandBetween(0, RGB(255, 255, 255))
End Function
```

Глава 31

Часто задаваемые вопросы о программировании в Excel

В этой главе...

- Списки часто задаваемых вопросов
- Общие вопросы об Excel
- Редактор Visual Basic
- Процедуры
- Функции
- Объекты, свойства, методы и события
- Пользовательские диалоговые окна
- Надстройки
- Пользовательский интерфейс

В этой главе приводятся ответы на часто задаваемые вопросы, касающиеся программирования в Excel.

Списки часто задаваемых вопросов

Те пользователи, которые часто "блуждают" в Интернете, знают, что такое FAQ (Frequently Asked Questions). Это список *часто задаваемых вопросов* (и ответов на них), касающихся определенной темы. Такие списки публикуются в конференциях, целью которых является сокращение потока одинаковых вопросов на определенную тему. Однако зачастую такие документы не выполняют возложенную на них функцию, так как одинаковые вопросы продолжают повторяться.

И поскольку люди в большинстве своем задают одинаковые вопросы о программировании на VBA, я создал данный документ, который содержит ответы на часто задаваемые вопросы о программировании в Excel:

- особенности Excel, оказывающие влияние на работу;
- часто задаваемые вопросы о программировании в Excel;
- справка по VBE.

Представляемый документ не ответит на все возникающие вопросы, однако он поможет решить ряд распространенных задач и даст представление о принципах решения некоторых проблем.

Все задаваемые вопросы были объединены в несколько категорий. Каждый вопрос относится к одной из следующих тем:

- общие вопросы об Excel;
- вопросы о редакторе Visual Basic;
- вопросы о процедурах;
- вопросы о функциях;
- вопросы об объектах, свойствах, методах и событиях;
- вопросы о пользовательских формах;
- вопросы о надстройках;
- вопросы об изменении пользовательского интерфейса Excel.

В некоторых случаях используется достаточно произвольная классификация, так как вопрос с одинаковым успехом может принадлежать к нескольким категориям. Более того, вопросы в каждой категории излагаются в свободном порядке.

Кстати, большая часть информации, приведенной в этой главе, подробно рассматривается в других главах книги.

Что делать, если в главе не найдется ответа на ваш вопрос

Обратитесь к предметному указателю. В книге приведено много информации, которую вряд ли можно отнести к категории ответов на часто задаваемые вопросы. Если ответ на интересующий вас вопрос в книге отсутствует, обратитесь к источникам, указанным в приложении А.

Общие вопросы об Excel

В этом разделе сведены ответы на вопросы общего характера относительно программирования в Excel.

Как записать макрос?

Щелкните на маленьком квадратном значке в левой части панели состояния.

Как запустить макрос на выполнение?

Для запуска макроса выберите команду Вид⇨Макросы⇨Макросы (View⇨Macros⇨Macros) или нажмите комбинацию клавиш <Alt+F8>. Либо выберите команду Разработчик⇨Код⇨Макросы (Developer⇨Code⇨Macros).

Что делать, если не отображается вкладка Разработчик?

Щелкните правой кнопкой мыши в области ленты и в контекстном меню выберите пункт Настройка ленты (Customize the Ribbon). На вкладке Настройка ленты (Customize Ribbon) в диалоговом окне Параметры Excel (Excel Options) установите флажок Разработчик (Developer) (в списке Основные вкладки (Main tabs)).

Я создал макрос и сохранил мою рабочую книгу. После повторного открытия книги макрос исчез. Что произошло?

При первом сохранении новой рабочей книги по умолчанию Excel предлагает отказаться от созданного вами макроса. Поэтому при сохранении файла внимательно читайте отображаемое на экране предостережение Excel и не щелкайте на кнопке Да (Yes). Если рабочая книга содержит макрос, сохраняйте ее в формате XLSM, а не XLSX.

Как можно скрыть ленту для освобождения места на экране?

В Excel 2010 появилась новая пиктограмма Свернуть ленту (Minimize the Ribbon), которая находится слева от пиктограммы Справка (Help) в строке заголовка. Щелкните на этой пиктограмме для того, чтобы скрыть/отобразить ленту. Для выполнения этой задачи можно также воспользоваться комбинацией клавиш <Ctrl+F1>. Чтобы отобразить/скрыть ленту с помощью VBA, воспользуйтесь методом Sendkeys.

```
Sub ToggleRibbon()
   Application.SendKeys "^{F1}"
End Sub
```

Для полного удаления ленты воспользуйтесь следующим XLM-макросом:

```
ExecuteExcel4Macro "SHOW.TOOLBAR(""Ribbon"",False)"
```

После выполнения этого оператора лента исчезнет из вашего поля зрения, и единственный способ вернуть ее обратно — выполнить этот оператор снова, причем последнему аргументу присваивается значение True.

Где же мои старые настраиваемые панели инструментов?

Выберите вкладку Надстройки (Add-Ins), после чего настраиваемые панели инструментов появятся в группе Настраиваемые панели инструментов (Custom Toolbars).

Можно ли сделать настраиваемые панели инструментов "плавающими"?

Нет, это невозможно. Старые настраиваемые панели инструментов зафиксированы на одном месте и находятся в группе Настраиваемые панели инструментов (Custom Toolbars), находящейся на вкладке Надстройки (Add-Ins).

Каким образом можно скрыть строку состояния в Excel 2010?

Для сокрытия строки состояния можно воспользоваться следующим кодом VBA:

```
Application.DisplayStatusBar = False
```

Существует ли утилита, которая преобразует мое приложение Excel в отдельный файл EXE?

Нет.

Почему комбинация клавиш <Ctrl+A> не выделяет все ячейки в рабочем листе?

Причиной этого может быть размещение указателя ячейки внутри таблицы. Если активная ячейка находится в таблице, нажмите комбинацию клавиш <Ctrl+A> трижды для выделения всех ячеек рабочего листа. После первого нажатия выделяются ячейки с данными, после второго — ячейки с данными и строка заголовка, а после третьего — все ячейки рабочего листа.

Почему команда Представления недоступна?

Причина этого заключается в том, что рабочая книга содержит таблицу. Преобразуйте таблицу в диапазон, затем воспользуйтесь командой Вид⇨Режимы просмотра книги⇨Представления (Views⇨Workbook Views⇨Custom Views).

Как добавить раскрывающийся список в ячейку, чтобы пользователь мог выбрать значение из списка?

Для решения этой задачи макрос не потребуется. Просто введите список корректных записей в единственный столбец. При желании можно скрыть этот столбец от пользователя. Выделите ячейку (или ячейки), в которой будет отображаться список, затем выберите команду Данные⇨Работа с данными⇨Проверка данных (Data⇨Data Tools⇨Data Validation), после чего выберите вкладку Параметры (Settings) в диалоговом окне Проверка вводимых значений (Data Validation). В раскрывающемся списке Тип данных (Allow) выберите опцию Список (List). В поле Источник (Source) укажите адрес диапазона или ссылку на одноколоночный список, находящийся на рабочем листе. Убедитесь в том, что флажок Распространить изменения на другие ячейки с тем же условием (In-Cell Dropdown) установлен. Если список короткий, просто введите его элементы, разделив их запятыми.

Могу ли я использовать метод раскрывающегося списка, если мой список находится на другом рабочем листе книги?

Да. В предыдущих версиях Excel требовалось создание имени списка (например, `ListEntries`). В Excel 2010 допускается применение диапазона в любом рабочем листе, даже находящемся в иной рабочей книге.

Будет ли свойство `Application.Calculation` влиять на все рабочие книги либо только на активную книгу?

Свойство `Calculation` является членом объекта `Application`. Поэтому выбранный режим вычислений будет применен во всех рабочих книгах. Невозможно выбрать режим вычислений для одной рабочей книги. В Excel 2000 и более поздних версиях появилось новое свойство объекта `Worksheet` под названием `EnableCalculation`. Если значение этого свойства `False`, рабочий лист не будет вычисляться, даже если вычисления инициируются пользователем. Для выполнения вычислений присвойте свойству `EnableCalculation` значение `True`.

Почему клавиша <F4> не может применяться для повторения ранее выполненных операций?

Я не знаю ответа на этот вопрос. К сожалению, очень полезная (в прошлом) клавиша <F4> утратила свое значение в Excel 2007 и более новых версиях. Например, если щелкнуть на значке Вставить лист (Insert Worksheet) и затем нажать клавишу <F4>, Excel не

повторит команду Вставить лист (Insert Worksheet). Но если для вставки листа воспользоваться комбинацией клавиш <Shift+F11>, клавиша <F4> повторит вставку листа.

Еще один пример. Если применить стиль к диаграмме с помощью команды Работа с диаграммами⇨Конструктор⇨Стили диаграмм (Chart Tools⇨Design⇨Chart Styles), эту команду невозможно применить к другой диаграмме, нажав клавишу <F4>. Возможно, эта проблема будет устранена в следующей версии Excel.

Что случилось со свойством "проговаривания" содержимого ячеек?

Для использования соответствующих команд следует настроить панель быстрого доступа либо ленту. Для выполнения этих задач воспользуйтесь диалоговым окном Параметры Excel (Excel Options). Речевые команды находятся в категории Команды не на ленте (Commands Not in the Ribbon). Все эти команды начинаются словом "Проговорить".

Я открыл рабочую книгу и увидел, что в ней только 65546 строк. Что случилось?

Рабочие листы Excel 2010 содержат 1048576 строк и 16384 столбца. Если отображается меньшее количество строк и столбцов, значит, рабочая книга находится в режиме совместимости. Если в Excel открывается рабочая книга, которая была сохранена с применением файлового формата предыдущих версий, не происходит ее автоматическое преобразование в рабочую книгу Excel 2010. Эту операцию следует выполнить вручную. Сохраните рабочую книгу в файловом формате Excel 2010, закройте ее и снова откройте. После этого появятся дополнительные строки и столбцы.

Как в старой рабочей книге можно использовать новые шрифты?

Начиная с версии Excel 2007 появились стандартные шрифты, которые проще для восприятия и выглядят гораздо лучше, чем в предыдущих версиях. Сначала создайте пустую книгу, нажав комбинацию клавиш <Ctrl+N>. Активизируйте старую рабочую книгу, затем выберите вкладку Главная (Home). Прокрутите вниз вертикальную полосу прокрутки в коллекции Стили (Styles), затем выберите команду Объединить стили (Merge Styles). В диалоговом окне Объединение стилей (Merge Styles) щелкните мышью дважды на новой рабочей книге, созданной нажатием комбинации клавиш <Ctrl+N>. После этого старые стили будут заменены новыми. Но этот прием может применяться только к тем ячейкам, которые не были отформатированы с применением других атрибутов шрифтов. Например, ячейки, выделенные полужирным шрифтом, останутся без изменений.

Как перейти в режим предварительного просмотра печати?

В Excel 2010 режим предварительного просмотра печати открывается автоматически после выполнения команды Файл⇨Печать (File⇨Print). Также в этот режим можно перейти путем щелчка на пиктограмме Страничный (Page Layout), находящейся в правой части строки состояния.

Для перехода в "классический" режим предварительного просмотра воспользуйтесь VBA. Следующий оператор осуществляет переход в режим предварительного режима для активного листа:

```
ActiveSheet.PrintPreview
```

При выборе нового шаблона документа рабочий лист не помещается на одной странице

Причина этой проблемы заключается в том, что в новой теме применяются новые шрифты. После применения темы выберите команду Разметка страницы⇨Темы⇨

Шрифты (Page Layout⇨Themes⇨Fonts), затем выберите исходные шрифты, которые будут применены в составе новой темы. Либо измените размер шрифта для стиля Обычный (Normal). Если компоновка рабочего листа имеет значение, выберите тему, прежде чем начнете работать над документом.

Как избавиться от штриховой линии разбиения страниц, которая отображается в режиме просмотра Обычный (Normal)?

Откройте диалоговое окно Параметры Excel (Excel Options), выберите вкладку Дополнительно (Advanced), затем перейдите в раздел Показать параметры для следующего листа (Display Options for This Worksheet) и отмените установку флажка Показывать разбиение на страницы (Show Page Breaks).

Могу ли я добавить на панель быстрого доступа либо на ленту параметр просмотра разбиения на страницы?

Нет. Эта весьма полезная команда не может быть добавлена на панель быстрого доступа либо ленту. Для отключения разбиения на страницы можно обратиться к следующему оператору VBA:

```
ActiveSheet.DisplayPageBreaks = False
```

Я попытался применить табличный стиль к таблице, но видимый эффект отсутствует. Что нужно сделать в подобной ситуации?

Это происходит в том случае, когда ячейки таблицы были отформатированы вручную. Выделите ячейки и выберите для цвета заливки параметр Нет заливки (No Fill), а для цвета шрифта — Авто (Automatic). После этого табличный стиль станет работоспособным.

Можно ли изменить цвет ярлычка листа?

Щелкните правой кнопкой мыши на ярлычке листа и выберите команду Цвет ярлычка (Tab Color). После этого цвет ярлычка изменится, создавая эффект применения другой темы документа.

Можно ли написать макрос VBA для воспроизведения звука?

Да, с помощью макроса можно воспроизводить файлы WAV и MIDI, но для этого следует обратиться к функциям Windows API (см. главу 11). В подобных случаях целесообразно воспользоваться объектом Speech. При вызове следующего оператора компьютер "здоровается" с пользователем:

```
Application.Speech.Speak ("Привет" & Application.UserName)
```

При открытии рабочей книги Excel предлагает обновить ссылки. Я просмотрел все формулы и не нашел ни одной ссылки. Это ошибка или нет?

Скорее всего, это не ошибка. Воспользуйтесь диалоговым окном Редактирование ссылок (Edit Links) (для открытия этого окна используется команда Файл⇨Сведения⇨Редактировать ссылки в файлах (File⇨Info⇨Edit Links to Files)). В диалоговом окне Редактирование ссылок (Edit Links) щелкните на кнопке Разрушить ссылку (Break Link). Обратите внимание на то, что ссылки могут находиться не только в формулах. Например, если в рабочей книге находится диаграмма, щелкайте на каждом ряде данных в диаграмме, просматривая функцию РЯД (SERIES), которая находится в строке формул. Если формула ссылается на другую книгу, это означает наличие ссылки. Для устранения

ссылки переместите данные диаграммы в текущую рабочую книгу, затем повторно создайте диаграмму.

Если рабочая книга содержит диалоговые листы Excel 5/95, выделяйте объекты в каждом диалоговом окне, проверяя при этом содержимое строки формул. Если объект содержит ссылку на другую рабочую книгу, измените или удалите ее.

Выберите команду Формулы⇨Определенные имена⇨Диспетчер имен (Formulas⇨Defined Name⇨Name Manager). Прокрутите вниз список в окне диспетчера имен и просмотрите столбец Диапазон (Refers To). Удалите имена, которые ссылаются на другую рабочую книгу или содержат некорректные ссылки (например, `#ССЫЛ!`). Именно по этой причине чаще всего возникают "фантомные ссылки".

Почему Excel сбоит при каждом запуске?

При запуске Excel открывается файл `*.xlb`, который включает настройки меню и панели инструментов. Если этот файл поврежден, это приведет к сбою при запуске Excel. В силу ряда причин размер этого файла может быть чрезвычайно большим. Результатом является очередной сбой при запуске Excel. Обычно размер файла `*.xlb` составляет 100 Кбайт или менее.

Если при каждом запуске Excel происходит сбой программы, попробуйте удалить файл `*.xlb`. Для этого закройте Excel и найдите на жестком диске файл `*.xlb`. (Имя файла и его местоположение варьируются.) Создайте резервную копию этого файла, удалите исходный файл и попробуйте перезапустить Excel. Обычно Excel запускается в штатном режиме, создавая новый файл `*.xlb`.

Удаление файла `*.xlb` также влечет за собой удаление всех настроек меню и панелей инструментов, которые находятся на вкладке Надстройки (Add-Ins).

Где можно найти примеры кода VBA?

В Интернете можно найти буквально тысячи примеров кода VBA. Для начала можете посетить мой веб-сайт, находящийся по следующему адресу:

```
http://spreadsheetpage.com
```

Можно попробовать поискать нужные сведения на сайте Google:

```
http://google.com
```

Редактор Visual Basic

Можно ли использовать функцию записи макросов VBA для создания всех макросов?

Нет. Запись макросов подходит только для очень простых операций. Макросы, которые задействуют переменные, циклы или прерывают поток выполнения операций, записать невозможно. Но вы вправе воспользоваться командой записи макросов для создания фрагментов кода, а также для получения информации о необходимых свойствах и методах.

Я создал универсальные макросы, которые должны быть доступны все время. Как обеспечить постоянный доступ к ним?

Можно сохранить макросы в персональной книге макросов. Это (обычно) скрытая рабочая книга, которая загружается в Excel автоматически. После записи макроса у вас появляется возможность сохранить его в персональной книге макросов. Соответствующий этой книге файл под названием `Personal.xlsb` находится в папке `\XLStart`.

Я не могу найти свою персональную книгу макросов. Где она?

Файл `Personal.xlsb` не будет существовать до тех пор, пока в него не записан хотя бы один макрос.

Я заблокировал проект VBA с помощью пароля и забыл пароль. Существует ли способ разблокировать проект?

Существует несколько программ взлома паролей, которые предоставляются независимыми производителями. Для их поиска можно воспользоваться одним из поисковых средств Интернета. В поиске вам помогут ключевые слова "пароль Excel". Существование подобных программ говорит о том, что пароли Excel не настолько надежны, как того хотелось бы.

Можно ли записать макрос, предназначенный для изменения пароля проекта?

Это невозможно. Средства защиты проекта VBA не предоставляются в объектной модели. Скорее всего, такой шаг сделан для усложнения работы программного обеспечения по взлому паролей.

Добавляемый новый модуль всегда начинается со строки `Option Explicit`. Что она означает?

Если строка `Option Explicit` находится в начале модуля, это означает, что необходимо объявлять все переменные, которые будут использоваться в пределах данного модуля (достаточно неплохая идея). Если требуется, чтобы эта строка не добавлялась в новые модули, запустите VBE и выберите команду Tools⇨Options (Сервис⇨Параметры). Перейдите на вкладку Editor (Редактор) и отмените установку флажка Require Variable Declaration (Явное описание переменных). После этого можно объявить переменные или переложить задачу определения типа данных на плечи интерпретатора VBA.

Почему мой код VBA отображается другим цветом? Можно ли изменить эти цвета?

В VBA цвета используются для выделения различных текстов в коде: комментариев, ключевых слов, идентификаторов, операторов с синтаксическими ошибками и т.д. Эти цвета можно переопределять, кроме того, допускается изменять шрифты, которые используются для выделения, и т.д. Воспользуйтесь командой Tools⇨Options (и перейдите на вкладку Editor Format (Формат редактора)) в редакторе VBE.

Можно ли удалить модуль VBA с помощью кода VBA?

Да. Следующий код удаляет модуль `Module1` из активной рабочей книги.

```
With ActiveWorkbook.VBProject
   .VBComponents.Remove .VBComponents("Module1")
End With
```

Этот код может не работать в вашей версии Excel. См. следующий вопрос.

Я написал макрос, который добавляет код VBA в проект VB. При запуске в среде Excel появляется сообщение об ошибке. Что здесь неправильно?

В Excel 2002 появилась новая настройка: Доверять доступ к объектной модели проектов VBA. По умолчанию эта настройка отключена. Для ее изменения выберите команду Файл⇨Параметры Excel⇨Центр управления безопасностью (File⇨Excel Options⇨Trust Center). Щелкните на кнопке Параметры центра управления безопасностью (Trust Center Settings) для отображения диалогового окна Центр управления

безопасностью (Trust Center). Щелкните на вкладке Параметры макросов (Macro Settings) и установите флажок Доверять доступ к объектной модели проектов VBA (Trust Access to the VBA Project Object Model).

Как изменить параметры безопасности макросов для пользователя? Как отключить появление сообщения "эта рабочая книга содержит макросы", когда открывается приложение?

Изменение уровня безопасности с помощью кода VBA приведет к полной бесполезности всей системы безопасности макросов. Подумайте над этим.

Как работает опция `UserInterfaceOnly` при защите рабочего листа?

При организации защиты рабочего листа с помощью кода VBA можно использовать следующий оператор:

```
ActiveSheet.Protect UserInterfaceOnly:=True
```

После выполнения этого оператора лист будет защищен, хотя и может изменяться с помощью макроса. Важно понимать, что эта установка не сохраняется вместе с рабочей книгой. При повторном открытии рабочей книги следует опять выполнить оператор для повторной установки `UserInterfaceOnly`.

Как узнать, находится ли в рабочей книге макровирус?

В редакторе VBE активизируйте проект, который соответствует рабочей книге. Просмотрите все модули кода и обратите внимание на код VBA, который вам незнаком. Обычно код вируса плохо форматирован и содержит большое количество переменных со странными названиями. Еще одной возможностью является использование коммерческих программ для поиска вирусов.

При использовании оператора & появляется сообщение об ошибке

Скорее всего, VBA интерпретирует амперсанд как символ объявления типа. Удостоверьтесь, что перед оператором конкатенации и после него добавлены пробелы.

Не работает символ продолжения строки в VBA (подчеркивания)

Для продолжения строки используются два символа: пробел и после него символ подчеркивания.

Я распространил приложение Excel для многих пользователей. На некоторых компьютерах процедуры обработки VBA не работают. Почему?

Процедуры обработки ошибок не работают, если была выбрана опция Break on All Errors (Останавливаться при любой ошибке). Эта опция находится в диалоговом окне Options (Параметры) вкладки General (Общие) в окне VBE. К сожалению, эту настройку нельзя изменить средствами VBA.

Процедуры

Чем отличается процедура VBA от макроса?

Ничем. Термин *макрос* "пришел" из ранней эпохи развития электронных таблиц. Эти термины в настоящее время являются взаимозаменяемыми.

Что такое процедура?

Процедура — это набор операторов VBA, который можно вызвать по имени. Если операторы должны возвратить явный результат (например, значение) процедуре, которая их вызывала, то эта процедура называется *функцией* и объявляется как `Function`. В противном случае процедура объявляется как `Sub`.

Что такое переменный тип данных?

Переменные, тип данных которых явно не определен, получают переменный тип данных (`Variant`). При использовании таких переменных VBA автоматически изменяет тип переменной в соответствии с присвоенным значением. Это особенно полезно при получении значений из рабочего листа, так как заранее неизвестно, что содержится в ячейках. Рекомендуется явно объявлять тип переменной. Для этого используются операторы `Dim`, `Public` и `Private`. Применение переменных типов данных замедляет работу приложения и вызывает неэффективное использование ресурсов памяти.

Чем отличается массив переменного типа от массива значений переменного типа?

Ячейка памяти переменного типа может содержать любой тип данных: единственное значение или массив значений (массив переменного типа). Следующий код создает переменную, которая содержит массив, включающий три элемента.

```
Dim X As Variant
X = Array(30, 40, 50)
```

В обычном массиве могут находиться элементы определенного типа, включая нетипизированные элементы (имеющие переменный тип). Приведенный ниже оператор создает массив, который состоит из 3-х элементов переменного типа.

```
Dim X (0 To 2) As Variant
```

Несмотря на то что массив элементов переменного типа и массив, представленный переменной типа `Variant`, концептуально отличаются, способ доступа к элементам массива остается таким же.

Что такое символ определения типа?

В VBA можно добавить к имени переменной символ, который будет указывать тип этой переменной. Например, можно объявить переменную `MyVar`, имеющую тип целого числа (integer). Для этого к имени переменной добавляется символ `%`:

```
Dim MyVar%
```

Ниже приведен список символов определения типа, которые поддерживаются в VBA:

- `Integer` — `%`;
- `Long` — `&`;
- `Single` — `!`;
- `Double` — `#`;
- `Currency` — `@`;
- `String` — `$`.

Символы определения типа включены в целях обеспечения совместимости. Обычно для объявления переменных используются зарезервированные слова.

Требуется создать процедуру, которая автоматически выполняет форматирование, зависящее от вводимых данных. Например, если вводится значение, большее 0, цвет фона ячейки становится красным. Возможно ли это?

Конечно, и для этого не обязательно программировать. Воспользуйтесь командой Excel Условное форматирование (Conditional Formatting), доступ к которой открывается в раскрывающемся списке кнопки Условное форматирование (Conditional Formatting), находящейся в группе Стили (Styles) вкладки Главная (Home).

Команда условного форматирования весьма полезна, но хотелось бы выполнять при вводе данных в ячейки и другие операции

Рекомендуем воспользоваться событием `Change` объекта рабочего листа. Как только будут внесены изменения в какую-либо из ячеек, возникнет событие `Change`. Если модуль кода объекта `Sheet` (Лист) содержит процедуру `Worksheet_Change`, то при возникновении события `Change` она будет выполняться автоматически.

Какие другие типы событий можно контролировать?

Огромное количество! Интерактивное справочное руководство содержит полный список доступных событий.

Я создал процедуру обработки события (`Sub Workbook_Open`), но она не запускается при открытии рабочей книги. В чем ошибка?

Скорее всего, процедура находится не там, где должна быть. Процедуры обработки событий рабочей книги должны располагаться в модуле кода объекта `ThisWorkbook` (ЭтаКнига). Процедуры обработки событий рабочего листа должны находиться в модуле кода объекта `Sheet` (Лист), что отображается в окне Project редактора VBE.

Еще одна причина появления подобной проблемы заключается в том, что в Excel отключен запуск макросов. Проверьте соответствующие настройки в окне Центр управления безопасностью (Trust Center). Для доступа к этому окну используется диалоговое окно Параметры Excel (Excel Options).

Я могу создать процедуру обработки событий для определенной рабочей книги. Можно ли создать процедуру обработки событий для всех открытых рабочих книг?

Да, но для этого необходимо создать модуль класса. Подробная информация приводится в главе 19.

Используются ли в VBA те же математические и логические операторы, что и при создании формул в Excel?

Да. Кроме того, в VBA предоставляются дополнительные операторы, которые нельзя использовать в формулах рабочего листа. Эти операторы перечислены в следующей таблице.

Оператор	Выполняемые функции
`\`	Деление с остатком
`Eqv`	Возвращает значение `True`, если аргументы формулы одновременно равны `True` либо `False`
`Imp`	Поразрядная логическая импликация на основе двух аргументов (применяется относительно редко)
`Is`	Сравнение двух объектов
`Like`	Сравнение строк с поддержкой групповых символов
`Xor`	Возвращает `True` только в том случае, когда один из аргументов равен `True`

Как вызвать процедуру, которая находится в другой рабочей книге?

Для этого необходимо воспользоваться методом Run объекта Application. Представленный далее оператор вызывает процедуру Macro1, которая расположена в рабочей книге Personal.xlsb.

```
Run "Personal.xlsb!Macro1"
```

Кроме того, можно создать ссылку на другую рабочую книгу. Для этого воспользуйтесь командой Tools⇨References (Сервис⇨Ссылки) редактора VBA. После добавления ссылки можно вызывать процедуры из книги, на которую установлена ссылка, без указания имени этой книги.

Я воспользовался VBA для создания нескольких функций и хотел бы применить эти функции в качестве формул рабочего листа, но указывать имя рабочей книги перед названием функции неудобно. Существует ли способ обойти эту проблему?

Да. Преобразуйте рабочую книгу, которая содержит определения функций, в надстройку Excel (XLAM). После открытия надстройки определенные в ней функции можно вызывать, не указывая названия рабочей книги.

Кроме того, если вы создадите ссылку на рабочую книгу, которая содержит определенные пользователем функции, то функцию также можно будет использовать без указания названия рабочей книги. Чтобы создать ссылку, необходимо воспользоваться командой Tools⇨References (Сервис⇨Ссылки) редактора VBE.

Возможно ли, чтобы определенная рабочая книга загружалась каждый раз, когда запускается Excel? Желательно также, чтобы при запуске Excel автоматически выполнялся макрос из этой рабочей книги

Почему бы и нет? Для того чтобы открыть рабочую книгу автоматически, сохраните ее в папке \XLStart. Чтобы автоматически запускать макрос из этой рабочей книги, создайте процедуру Workbook_Open в модуле кода объекта ThisWorkbook (ЭтаКнига).

У меня есть рабочая книга, в которой находится процедура Workbook_Open. Существует ли способ предотвратить автоматический запуск этой процедуры при открытии рабочей книги?

Да. Удерживайте клавишу <Shift> при выборе команды Файл⇨Открыть (File⇨Open). Для того чтобы предотвратить выполнение процедуры Workbook_BeforeClose, удерживайте клавишу <Shift> при закрытии рабочей книги. Использование клавиши <Shift> не позволяет предотвратить запуск этих процедур при открытии надстройки.

Может ли процедура VBA получить доступ к значению ячейки в неоткрытой рабочей книге?

VBA не в состоянии это сделать, но вы сможете достичь поставленной цели с помощью языка XLM. К счастью, XLM можно вызывать из VBA. Ниже приведен простой пример получения значения ячейки A1 из листа Лист1 рабочей книги myfile.xlsx, которая находится в папке c:\files.

```
MsgBox ExecuteExcel4Macro("'c:\files\[myfile.xlsx]Лист1'!R1C1")
```

Обратите внимание на то, что ячейка идентифицируется в формате R1C1.

Как предотвратить отображение сообщения о необходимости сохранения рабочей книги при ее закрытии с помощью VBA?

Можно воспользоваться следующим оператором:

```
ActiveWorkbook.Close SaveChanges:=False
```

Вы также вправе установить значение свойства `Saved` объекта `Workbook` равным `True`. Для этого воспользуйтесь таким оператором:

```
ActiveWorkbook.Saved = True
```

При его выполнении файл не сохраняется, поэтому после закрытия рабочей книги все несохраненные изменения будут утеряны.

Более универсальным решением по отмене сообщений в Excel будет использование следующего оператора:

```
Application.DisplayAlerts = False
```

Обычно после закрытия файла свойству `DisplayAlerts` присваивается значение `True`.

Как сделать так, чтобы макрос запускался регулярно в определенное время?

Для этого необходимо воспользоваться методом `OnTime` объекта `Application`. Это позволяет указать процедуру, которая будет выполняться в определенное время суток. Когда процедура завершает свою работу, необходимо еще раз воспользоваться методом `OnTime`, чтобы запланировать следующий вызов.

Как сделать так, чтобы макрос не отображался в списке макросов?

Для предотвращения отображения макроса в диалоговом окне Макрос (Macro) (отображается в результате выполнения команды Вид⇨Макросы⇨Макросы (View⇨Macros⇨Macro)) объявите процедуру с помощью ключевого слова `Private`.

```
Private Sub MyMacro()
```

Также можно добавить необязательный аргумент определенного типа данных.

```
Sub MyMacro (Optional FakeArg as Long)
```

Существует ли возможность сохранения диаграммы в файле формата GIF?

Да. Приведенный ниже код сохраняет первую встроенную диаграмму рабочего листа `Лист1` в виде файла формата GIF с названием `Mychart.gif`.

```
Set CurrentChart = Sheets("Лист1").ChartObjects(1).Chart
Fname = ThisWorkbook.Path & "\Mychart.gif"
CurrentChart.Export Filename:=Fname, FilterName:="GIF"
```

Доступны ли переменные одной процедуры VBA другим процедурам VBA? Что если процедура находится в другом модуле или даже в другой рабочей книге?

В данном случае речь идет об области действия переменной. Существует три типа действия: локальная, в пределах модуля и общедоступная (глобальная). Локальные переменные имеют самую узкую область действия и объявляются внутри процедуры. Локальная переменная используется только в пределах процедуры, в которой она объявлена. Переменные с областью действия уровня модуля объявляются в начале модуля, до объявления первой процедуры. Переменные с областью действия уровня модуля используются во всех процедурах, которые определены в этом модуле. Общедоступные переменные имеют самую широкую область действия и объявляются с помощью ключевого слова `Public`.

Функции

Я создал функцию VBA для использования в формулах рабочего листа. Но в результате ее выполнения отображается сообщение об ошибке #ИМЯ?. Что здесь неправильно?

Скорее всего, причина в том, что функция находится в модуле кода листа (например, `Лист1`) либо в модуле `ThisWorkbook` (ЭтаКнига). Пользовательские функции рабочего листа должны находиться в стандартных модулях VBA.

Я создал функцию VBA, которая работает при вызове ее из другой процедуры, а при использовании в составе формулы рабочего листа не работает. В чем здесь проблема?

Функции VBA, вызываемые в формулах рабочего листа, ограничены в своих возможностях. В общем случае эти функции являются строго пассивными. Это означает, что функции не могут изменять активную ячейку, применять форматирование, открывать рабочую книгу или изменять активный лист. При попытке выполнения любого из этих действий возвращается ошибка.

Я создал пользовательскую функцию рабочего листа. При получении доступа к этой функции из диалогового окна мастера функций выводится сообщение о невозможности предоставления справочной информации. Как устранить эту проблему?

Для того чтобы добавить к функции описание, активизируйте рабочую книгу, которая содержит эту функцию. Затем выберите команду Вид⇨Макросы⇨Макросы (View⇨Macros⇨Macros) для отображения диалогового окна Макрос (Macro). Функция не отображается в списке, поэтому укажите ее имя в поле Имя макроса (Macro Name). После ввода имени функции щелкните на кнопке Параметры (Options) для отображения диалогового окна Параметры макроса (Macro Options). В поле Описание (Description) введите описание функции.

Можно ли отображать справочную информацию для аргументов функции в диалоговом окне мастера функций?

Да. В Excel 2010 появился новый аргумент метода `MacroOptions`, позволяющий реализовать эту возможность. Можно написать макрос, выполняющий присваивание описаний аргументам функции. Дополнительные сведения по этой теме можно найти в главе 10.

Моя функция рабочего листа отображается в категории "Определенные пользователем" окна мастера функций. Как переместить функцию в другую категорию?

Для этого необходимо воспользоваться кодом VBA. Следующий оператор перемещает функцию `MyFunc` в категорию 1 (Финансовые функции (Financial)):

```
Application.MacroOptions Macro:="MyFunc", Category:=1
```

Представленная ниже таблица содержит список допустимых номеров категорий функций.

Номер	Категория
0	Без категории (отображается только в категории Все функции)
1	Финансовые
2	Дата и время
3	Математические и тригонометрические
4	Статистические

Окончание таблицы

Номер	Категория
5	Ссылки и массивы
6	Работа с базами данных
7	Текстовые
8	Логические
9	Информационные
10	Команды (обычно эта категория скрыта)
11	Настройка (обычно эта категория скрыта)
12	Управление макросами (обычно эта категория скрыта)
13	DDE/внешние (обычно эта категория скрыта)
14	Определенные пользователем (по умолчанию)
15	Инженерные

Как создать новую категорию функций?

Укажите текстовую строку для аргумента `Category` метода `MacroOptions`. Ниже приводится пример.

```
Application.MacroOptions Macro:="MyFunc", _
  Category:="Корпоративные функции XYZ"
```

У меня есть пользовательская функция, которая будет применяться в формуле рабочего листа. Как сделать так, чтобы при вводе неверного значения функция возвращала значение ошибки (#ЗНАЧ!)?

Если функция называется `MyFunction`, то можно использовать следующий оператор для возврата кода ошибки в ячейку, содержащую функцию:

```
MyFunction = CVErr(xlErrValue)
```

В этом примере `xlErrValue` является предопределенной константой. Константы остальных кодов ошибок описываются в интерактивном справочном руководстве.

В коде VBA я воспользовался функцией Windows API, которая работала великолепно. Затем я передал рабочую книгу коллеге, он попытался открыть ее и получил сообщение об ошибке компиляции. В чем причина появления этой проблемы?

Вероятнее всего, причина появления проблемы заключается в том, что ваш коллега использует 64-разрядную версию Excel 2010. Для обеспечения совместимости с этой версией объявления API-функций должны включать ключевое слово "PtrSafe". Пример объявления API-функции, которое корректно в 32-разрядной версии Excel, но вызывает ошибку в 64-разрядной версии Excel 2010, приводится ниже.

```
Declare Function GetWindowsDirectoryA Lib "kernel32" _
   (ByVal lpBuffer As String, ByVal nSize As Long) As Long
```

Во многих случаях обеспечение совместимости объявления с 64-разрядной версией Excel достигается путем добавления слова "PtrSafe" после ключевого слова `Declare`. Этот метод обычно применяется при объявлении API-функций, хотя некоторые такие функции требуют изменения типа данных аргументов.

Ниже приводится пример объявления, которое совместимо как с 32-, так и с 64-разрядной версиями Excel 2010.

```
Declare PtrSafe Function GetWindowsDirectoryA Lib "kernel32" _
   (ByVal lpBuffer As String, ByVal nSize As Long) As Long
```

Причина сбоя кода VBA в Excel 2007 (и более ранних версиях) может заключаться в том, что ключевое слово `PtrSafe` не распознается. Ниже представлен пример использования директив компилятора для объявления API-функции, которая совместима с 32-разрядной версией Excel (включая версии, предшествующие Excel 2010) и с 64-разрядной версией Excel 2010.

```
#If VBA7 And Win64 Then
   Declare PtrSafe Function GetWindowsDirectoryA Lib "kernel32" _
      (ByVal lpBuffer As String, ByVal nSize As Long) As Long
#Else
   Declare Function GetWindowsDirectoryA Lib "kernel32" _
      (ByVal lpBuffer As String, ByVal nSize As Long) As Long
#End If
```

Первый оператор `Declare` применяется в том случае, когда значения констант `VBA7` и `Wind64` будут `True` (в случае использования 16-разрядной версии Excel 2010). Для всех других версий Excel применяется второй оператор `Declare`.

Как выполнить пересчет формул, применяющих пользовательскую функцию?

Для этого можно воспользоваться комбинацией клавиш <Ctrl+Alt+F9>.

Можно ли использовать встроенные функции рабочего листа Excel в коде VBA?

В большинстве случаев — да. Функции Excel рабочего листа задаются с помощью метода `WorksheetFunction` объекта `Application`. Например, доступ к функции `SUM` (`СУММ`) вы получите, введя следующий оператор:

```
Ans = Application.WorksheetFunction.Sum(Range("A1:A3"))
```

В этом примере подсчитывается сумма всех значений в ячейках A1:A3 (в активном листе), которая присваивается переменной `Ans`.

Если VBA содержит эквивалент функции рабочего листа, то последнюю нельзя использовать на рабочем листе. Например, если VBA задает функцию подсчета квадратного корня (`Sqr`), то функцию `SQRT` рабочего листа в коде VBA использовать нельзя.

Существует ли способ задать разрыв строки в тексте окна сообщения?

Для этого необходимо воспользоваться символом возврата каретки или перевода строки. Представленный далее оператор отображает окно сообщения, которое содержит текст, состоящий из двух строк. Встроенная константа `vbNewLine` представляет символ перевода строки.

```
MsgBox "Привет" & vbNewLine & Application.UserName
```

Объекты, свойства, методы и события

Существует ли список событий Excel, которые можно использовать?

Да. Соответствующие сведения можно найти в справочной системе.

Количество доступных свойств и методов огромно. Как воспользоваться мѐтодами и свойствами, доступными для определенного объекта?

Существует несколько способов получения этой информации. Можно воспользоваться окном Object Browser, которое отображается в редакторе VBE. Нажмите клавишу

<F2> для получения доступа к окну Object Browser и выберите Excel из раскрывающегося списка Libraries/Workbooks (Все библиотеки). Список Classes (Классы) (слева) содержит все доступные объекты Excel. При выборе объекта его свойства и методы отображаются справа в списке Members (Компоненты).

Кроме того, список свойств и методов можно отобразить при введении имени объекта. Например, введите следующий код:

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

При вводе точки будет отображен список всех доступных свойств и методов объекта Range. Если этого не произошло, то необходимо выбрать Tools⇨Options (Сервис⇨ Параметры), перейти на вкладку Editor (Редактор) и установить флажок Auto List Members (Автоматическая вставка объектов). К сожалению, свойство автоперечисления работает далеко не со всеми объектами. Например, ввод следующего оператора не приведет к отображению списка всех свойств и методов:

```
ActiveSheet.Shapes(1).
```

Большое количество информации по языку VBA содержит также справка, которую можно вызвать в окне справки Visual Basic Editor. В нем вы найдете описание методов и свойств всех объектов, которые могут вызвать интерес разработчика. Самым простым способом получения этой информации является ввод имени объекта в окне отладки (Immediate) в нижней части окна VBE или наведение указателя на имя объекта с последующим нажатием клавиши <F1>. В результате будет отображен раздел справочного руководства, посвященный этому объекту.

В чем смысл использования коллекций? Является ли коллекция объектом? Что такое коллекция?

Коллекция — это объект, который содержит группу подобных объектов. Коллекция получает название в виде существительного во множественном числе. Например, коллекция Worksheets — это объект, который содержит объекты Worksheet рабочей книги. Коллекцию можно воспринимать как массив: Worksheets(1) указывает первый объект Worksheet в коллекции Workbooks. Вместо индекса допускается указывать имя рабочего листа, например Worksheets("Лист1"). Концепция коллекции упрощает одновременное управление подобными объектами. А для циклического перебора членов коллекции можно воспользоваться конструкцией For Each-Next.

Когда в коде VBA я ссылаюсь на рабочий лист, возникает сообщение об ошибке "subscript out of range". Но я не использую подсценарии. В чем тут дело?

Эта ошибка возникает при попытке получить доступ к несуществующему члену коллекции. Например, приведенный ниже оператор создает сообщение об ошибке, если активная рабочая книга не содержит рабочий лист, который называется MySheet.

```
Set X = ActiveWorkbook.Worksheets("MySheet")
```

Как лишить пользователя возможности прокручивать рабочий лист?

Для этого можно или скрыть неиспользуемые строки и столбцы, или воспользоваться оператором VBA для перенастройки полосы прокрутки на рабочем листе. Следующий оператор устанавливает область прокрутки на листе Лист1 таким образом, что пользователь не может активизировать ячейки за пределами диапазона B2:D50.

```
Worksheets("Лист1").ScrollArea = "B2:D50"
```

Для того чтобы восстановить первоначальную область прокрутки, воспользуйтесь таким оператором:

```
Worksheets("Лист1").ScrollArea = ""
```

Помните, что значение свойства `ScrollArea` не сохраняется вместе с рабочей книгой. Таким образом, необходимо устанавливать это свойство при каждом следующем открытии рабочей книги. Данный оператор можно разместить в процедуре обработчика событий `Workbook_Open`.

Какая разница между использованием методов `Select` и `Apllication.Goto`?

Метод `Select` объекта `Range` выделяет диапазон только на активном рабочем листе. Метод `Application.Goto` можно использовать для выделения диапазона на любом листе рабочей книги. Метод `Application.Goto` может либо активизировать, либо деактивизировать другой лист. Кроме того, метод `Goto` позволяет прокручивать рабочий лист, чтобы расположить выделенный диапазон в левом верхнем углу экрана.

Какая разница между активизацией и выделением диапазона?

В отдельных случаях методы `Activate` и `Select` приводят к одинаковому эффекту. Но зачастую они возвращают разные результаты. Предположим, что выделен диапазон A1:C3. Приведенный ниже оператор активизирует ячейку C3. Первоначальный диапазон остается выделенным, но ячейка C3 становится активной, т.е. курсор находится в этой ячейке.

```
Range("C3").Activate
```

Предположим также, что диапазон A1:C3 выделен. Представленный далее оператор выделяет единственную ячейку, которая становится активной.

```
Range("C3").Select
```

Существует ли быстрый способ удаления всех значений из рабочего листа без удаления формул?

Да. Код выполняется по отношению к активному рабочему листу и удаляет значения всех ячеек, которые не содержат формул (форматирование ячеек тоже не очищается).

```
On Error Resume Next
Cells.SpecialCells(xlCellTypeConstants, 23).ClearContents
```

Второй аргумент, 23, представляет собой сумму значений, представленных следующими встроенными константами: `xlErrors` (16), `xlLogical` (4), `xlNumbers` (1) и `xlTextValues` (2).

Использование оператора `On Error Resume Next` позволяет отменить сообщение об ошибке, которое возникает, если ни одна ячейка не соответствует указанному критерию.

Я знаю, как писать операторы VBA, выделяющие диапазон, для которого известен адрес. Объясните, как создать оператор выделения, если известны только номер строки и номер столбца?

Для этого воспользуйтесь методом `Cells`. Следующий оператор выделяет ячейку в пятой строке и двенадцатом столбце (т.е. ячейку L5):

```
Cells(5, 12).Select
```

При попытке создать макрос, вызывающий команду Excel Файл⇨Выход из Excel (File⇨Exit Excel), Excel закрывается до того, как отобразится сгенерированный код. Существует ли команда VBA для выхода из Excel?

Для выхода из Excel используйте следующую инструкцию:

```
Application.Quit
```

Как отключить обновление экрана в процессе работы макроса?

Представленный далее оператор отключает обновление экрана и ускоряет работу макроса, модифицирующего отображаемую информацию.

```
Application.ScreenUpdating = False
```

По завершении выполнения процедуры свойству `ScreenUpdating` присваивается значение `True`. Для восстановления обновления экрана воспользуйтесь следующим оператором:

```
Application.ScreenUpdating = False
```

Какой самый простой способ создания имени диапазона с помощью VBA?

Если включить запись макроса при создании имени диапазона, то будет получен следующий код.

```
Range("D14:G20").Select
ActiveWorkbook.Names.Add Name:="InputArea", _
RefersToR1C1:="=Лист1!R14C4:R20C7"
```

Но намного проще использовать следующий оператор:

```
Sheets("Лист1").Range("D14:G20").Name = "InputArea"
```

Как определить, имеет ли имя конкретная ячейка или диапазон?

Для этого необходимо проверить значение свойства `Name` объекта `Name`, который содержится в объекте `Range`. Следующая функция принимает диапазон в качестве аргумента и возвращает имя диапазона, если оно существует. Если диапазон не имеет имени, то функция возвращает значение `False`.

```
Function RangeName(rng) As Variant
   On Error Resume Next
   RangeName = rng.Name.Name
   If Err <> 0 Then RangeName = False
End Function
```

На первый взгляд создается впечатление, что в Excel 2010 отсутствует окно предварительного просмотра. Переход к предварительному просмотру осуществляется в режиме просмотра Backstage (после выполнения команды Файл⇨Печать), а классическое окно предварительного просмотра отсутствует

Единственный способ перехода в классическое окно предварительного просмотра — воспользоваться следующим оператором VBA:

```
ActiveSheet.PrintPreview
```

Можно ли отображать сообщения в строке состояния в процессе выполнения макроса? Я использую сложный макрос и неплохо бы иметь представление о ходе его выполнения

Рекомендуется присвоить описательный текст свойству `StatusBar` объекта `Application`. Приведем пример соответствующего оператора.

```
Application.StatusBar = "Обрабатывается файл " & FileNum
```

После завершения процедуры необходимо восстановить строку состояния с помощью такого фрагмента кода.

```
Application.StatusBar = False
Application.StatusBar = ""
```

Я написал макрос VBA, который копирует диапазон и вставляет его в другое место на рабочем листе. Этот макрос использует метод `Select`. Существует ли более эффективный способ копирования и вставки?

Да. Хотя при записи макроса ячейки сначала выделяются, выделение выполнять вовсе не обязательно – на самом деле это только замедляет работу макроса. Запись простой операции копирования и вставки приводит к генерированию кода VBA, состоящего из четырех строк. Две из них вызывают метод `Select`.

```
Range("A1").Select
Selection.Copy
Range("B1").Select
ActiveSheet.Paste
```

Эти четыре строки кода можно заменить одним оператором, который приводится ниже.

```
Range("A1").Copy Range("B1")
```

Обратите внимание на то, что последний оператор не требует использования метода `Select`.

Я не нашел метода сортировки массива VBA. Значит ли это, что нужно копировать значения на рабочий лист и применять метод `Range.Sort`?

В языке VBA не существует встроенных средств сортировки массивов. Копирование массива на рабочий лист — это только один из методов, но можно создать собственную процедуру сортировки. В настоящее время существует довольно много алгоритмов сортировки, часть из которых достаточно просто реализуется с помощью VBA. В этой книге приводится код VBA нескольких методов сортировки.

Мой макрос работает с выделенными ячейками, но он неудачно завершается, если выделенной оказывается диаграмма. Как удостовериться в том, что выделен диапазон ячеек?

Можно воспользоваться функцией VBA `TypeName`, которая применяется для проверки объекта `Selection`. Ниже приведен пример такого кода.

```
If TypeName(Selection) <> "Range" Then
   MsgBox "Выделите диапазон!"
   Exit Sub
End If
```

Еще одним вариантом является использование свойства `RangeSelection`, которое возвращает объект `Range`, представляющий выделенные ячейки рабочего листа. Данное свойство возвращает выделенные ячейки на рабочем листе указанного окна, даже если

помимо них выделен графический объект. Это свойство объекта Window, а не объекта Workbook. Следующий оператор отображает адрес выделенного диапазона:

```
MsgBox ActiveWindow.RangeSelection.Address
```

Как определить, активна ли диаграмма?

Используйте следующее выражение.

```
If ActiveChart Is Nothing Then
   MsgBox "Выделите диаграмму "
   Exit Sub
End If
```

Окно сообщения появится на экране только в том случае, если диаграмма неактивна (имеется в виду как внедренная диаграмма, так и диаграмма на отдельных листах).

Макрос VBA должен подсчитать количество строк, выделенных пользователем. Метод Selection.Rows.Count не применяется, если выделены несмежные строки. Это ошибка?

На самом деле так и должно быть. Метод Count возвращает количество элементов в первой выделенной области (предположим, что выделение состоит из нескольких областей). Для того чтобы получить точное количество строк, код VBA должен определить количество выделенных областей и подсчитать количество строк в каждой области. Сначала необходимо воспользоваться методом Selection.Area.Count, чтобы получить количество областей. Ниже приведен пример, который сохраняет общее количество выделенных строк в переменной NumRows.

```
NumRows = 0
For Each area In Selection.Areas
   NumRows = NumRows + area.Rows.Count
Next area
```

Кстати, это относится и к подсчету количества выделенных столбцов.

Я использую Excel для создания накладных. Существует ли способ генерации уникального номера накладной?

Одним из способов является использование системного реестра. Следующий код демонстрирует такой способ.

```
Counter = GetSetting("XYZ Corp", "InvoiceNum", "Count", 0)
Counter = Counter + 1
SaveSetting "XYZ Corp", "InvoiceNum", "Count", Counter
```

При выполнении данного кода текущее значение извлекается из системного реестра, увеличивается на единицу и назначается переменной Counter. После этого обновленное значение сохраняется в системном реестре. Значение переменной Counter можно использовать в качестве уникального номера накладной.

Этот прием можно применять и в других целях. Например, можно подсчитать количество раз, когда была открыта рабочая книга. Для этого код, подобный указанному выше, включается в процедуру Workbook_Open.

Существует ли свойство рабочей книги Excel, которое определяет ее постоянное отображение на экране?

К сожалению, не существует.

Существует ли оператор VBA для выделения последнего значения в строке или столбце? Как правило, я могу воспользоваться комбинацией клавиш <Ctrl+Shift+↓> или <Ctrl+Shift+→> для выполнения этого действия, но как это сделать с помощью макроса?

Эквивалентом комбинации клавиш <Ctrl+Shift+↓> в VBA выступает следующий оператор:

```
Selection.End(xlDown).Select
```

Для иных направлений используются три другие константы: `xlToLeft`, `xlToRight` и `xlUp`.

Как определить последнюю непустую ячейку в выбранном столбце?

Представленный ниже оператор отображает адрес последней непустой ячейки в столбце A.

```
MsgBox ActiveSheet.Cells(Rows.Count, 1).End(xlUp).Address
```

Но этот оператор не работает, если последняя ячейка в столбце непустая. Для того чтобы обойти эту маловероятную ситуацию, воспользуйтесь следующим кодом.

```
With ActiveSheet.Cells(Rows.Count, 1)
   If IsEmpty(.Value) Then
      MsgBox .End(xlUp).Address
   Else
      MsgBox .Address
   End If
End With
```

Ссылки VBA могут стать очень длинными, особенно если нужно точно указать объект в рабочей книге и рабочем листе. Существует ли способ сократить длину ссылок?

Да. Для этого воспользуйтесь оператором `Set` для создания объектной переменной. Ниже приведен пример подобного кода.

```
Dim MyRange as Range
Set MyRange = ThisWorkbook.Worksheets("Лист1").Range("A1")
```

После выполнения оператора `Set` можно ссылаться на объект `Range` по имени `MyRange`. Например, вы имеете возможность определить значение ячейки с помощью следующего оператора:

```
MyRange.Value = 10
```

Кроме упрощения ссылок на объекты, использование таких переменных ускоряет выполнение кода.

Существует ли способ объявления массива, если неизвестно, сколько элементов он будет содержать?

Да. Можно объявить динамический массив с помощью оператора `Dim`. При этом используются пустые скобки. Как только станет известно, сколько элементов будет содержаться в массиве, воспользуйтесь оператором `ReDim`, чтобы переопределить массив. Оператор `ReDim Preserve` позволяет переопределить массив без потери текущего содержимого.

Можно ли предоставить пользователю возможность отменить результат выполнения макроса?

Да, но эта возможность недоступна по умолчанию. Для того чтобы включить ее, модуль кода VBA должен отслеживать изменения, проведенные макросом. Макрос должен быть готов отменить внесенные изменения при выборе пользователем команды Отмена.

Чтобы разрешить пользователю выполнять команду Отмена, воспользуйтесь методом OnUndo в качестве последнего оператора макроса. Этот метод позволяет указать текст, который отображается в меню команды Отмена, а также процедуру, вызываемую командой Отмена. Ниже приводится пример.

```
Application.OnUndo "Последний макрос ", "MyUndoMacro"
```

Можно ли приостановить выполнение макроса, чтобы пользователь мог ввести данные в определенную ячейку?

Для того чтобы получить значение от пользователя и поместить его в определенную ячейку, можно воспользоваться функцией Excel InputBox. Например, первый оператор фрагмента кода, приведенного ниже, отображает окно ввода. Как только пользователь введет значение, оно будет помещено в ячейку A1.

```
UserVal = Application.InputBox(prompt:="Значение", Type:=1)
If TypeName(UserVal)<>"Boolean" Then Range("A1") = UserVal
```

VBA имеет функцию InputBox, но метод с таким же названием содержится в объекте Application. Они одинаковы?

Нет. Метод Excel InputBox является более гибким, так как позволяет проверить введенное пользователем значение. В предыдущем примере в качестве значения аргумента Type метода InputBox использовалась единица (представляет числовое значение). Это значение позволяет удостовериться, что пользователь ввел в поле числовое значение.

Для того чтобы написать оператор VBA, который создает формулу, следует вставить символ кавычек (") в текст. Как это сделать?

Предположим, что вам с помощью кода VBA необходимо ввести следующую формулу в ячейку с адресом B1.

```
=ЕСЛИ(A1="Да", ИСТИНА, ЛОЖЬ)
```

Следующий оператор генерирует сообщение о синтаксической ошибке, поскольку используются встроенные символы кавычек.

```
Range("B1").ormula = "=ЕСЛИ(A1="Да",ИСТИНА,ЛОЖЬ)" ' неверно
```

Решением будет последовательное использование двух двойных кавычек. В этом случае внутренние кавычки Excel интерпретирует как одинарные кавычки. Представленный далее оператор приводит к необходимому результату.

```
Range("B1").Formula = "=ЕСЛИ(A1=""Yes"",ИСТИНА,ЛОЖЬ)"
```

Еще одним способом является использование функции VBA Chr с аргументом 34. Функция возвращает символ кавычки. Следующий пример демонстрирует применение этого способа.

```
Range("B1").Formula =
   "=ЕСЛИ(A1=" & Chr(34) & "Да" & Chr(34) & ",ИСТИНА,ЛОЖЬ)"
```

Еще один способ заключается в том, чтобы воспользоваться формулой, в которой вместо символов кавычек используются апострофы. Затем с помощью функции VBA Replace апострофы заменяются кавычками.

```
MyFormula = "=ЕСЛИ A1='Да',TRUE,FALSE)"
Range("B1").Formula = Replace(MyFormula, "'", Chr(34))
```

Я создал массив, но первый его элемент воспринимается как второй. Что я делаю неправильно?

Если не указать обратного, VBA использует значение `0` в качестве индекса первого элемента массива. Если необходимо, чтобы индекс массива начинался с `1`, вставьте представленный ниже оператор в начало модуля VBA.

```
Option Base 1
```

Кроме того, можно указать верхнюю и нижнюю границы массива при его создании. Ниже приводится пример.

```
Dim Months(1 To 12) As String
```

Я слышал о том, что некоторые вещи в Excel можно запрограммировать только с помощью устаревшего макроязыка XLM. Это правда?

Нет, это не вполне соответствует действительности. Одна из задач, которые ставились перед разработчиками Excel 2010, — устранение ограничений на возможности языка VBA.

Например, в предыдущих версиях Excel требовались макросы XLM для определения описаний аргументов в пользовательских функциях рабочего листа. В Excel 2010 эту задачу можно выполнить с помощью нового аргумента `ArgumentDescriptions` метода `MacroOptions`. Еще один пример нового свойства — коллекция `AddIns2`, которая позволяет использовать все открытые надстройки (не только те, которые установлены). В прежних версиях Excel доступ к открытым (но не установленным) надстройкам требовал макроса XLM.

Как добиться от кода VBA максимального быстродействия?

Предложим несколько общих советов.

- Удостоверьтесь, что объявлены все используемые переменные. Воспользуйтесь оператором `Option Explicit` в начале модуля, чтобы сделать объявление переменных обязательным.
- Если ссылка на объект Excel применяется более одного раза, создайте для этого объекта переменную.
- Используйте конструкцию `With-End With` везде, где это возможно.
- Если макрос записывает информацию на рабочий лист, отключите обновление экрана с помощью оператора `Application.ScreenUpdating = False`.
- Если приложение вводит данные в ячейки, на которые ссылаются несколько формул, то следует включить ручной режим пересчета, чтобы избежать лишних вычислений.

Пользовательские диалоговые окна

Необходимо получить от пользователя данные, но диалоговое окно `UserForm` требует сложного программирования. Существует ли альтернатива?

Да. Обратитесь к функциям VBA `MsgBox` и `InputBox`. Кроме того, можно воспользоваться методом Excel `InputBox`.

Допустим, что в диалоговом окне UserForm находится 12 элементов управления CommandButton. Как создать единый макрос, который будет вызываться при щелчке на любой из кнопок?

Для решения этой задачи не существует простого способа, так как каждый элемент управления CommandButton имеет собственную процедуру Click. Одним из решений является вызов дополнительной процедуры из каждой процедуры обработки события CommandButton_Click. Другое решение — это создание модуля класса для получения нового объекта (см. главу 15).

Существует ли способ отобразить диаграмму в диалоговом окне UserForm?

Непосредственного способа получения такой диаграммы нет. Одним из решений является использование макроса, который сохраняет диаграмму в виде файла формата GIF и загружает файл GIF в элемент управления Image.

Как удалить кнопку "×" из строки заголовка диалогового окна UserForm, с тем чтобы лишить пользователя возможности закрывать диалоговое окно с помощью этой кнопки

Удаление кнопки "×" (Закрыть) из строки заголовка диалогового окна UserForm требует использования нескольких сложных функций Windows API. Более простым решением будет перехват всех попыток закрыть диалоговое окно. Воспользуйтесь процедурой UserForm_QueryClose, которая находится в модуле кода диалогового окна UserForm. Следующий пример иллюстрирует результат того, что пользователь не может закрыть диалоговое окно с помощью кнопки "×".

```
Private Sub UserForm_QueryClose _
   (Cancel As Integer, CloseMode As Integer)
      If CloseMode = vbFormControlMenu Then
         MsgBox "Вы не сможете закрыть диалоговое окно."
         Cancel = True
      End If
End Sub
```

Элементы созданного мною диалогового окна UserForm связаны с ячейками рабочего листа с помощью свойства ControlSource. Существует ли более удачный способ получения подобного результата?

Рекомендуется избегать использования ссылок на ячейки рабочего листа, кроме случаев, когда это действительно необходимо. Создание таких ссылок замедляет работу приложения, так как рабочий лист пересчитывается каждый раз, когда элемент управления вносит изменения в связанную ячейку. Кроме того, если в диалоговом окне имеется кнопка Отмена, может оказаться, что перед щелчком на этой кнопке значения в ячейках уже будут изменены.

Существует ли способ создания массива элементов управления для диалогового окна UserForm? Это можно сделать в Visual Basic, но я не могу понять, как это делается в VBA

Массив элементов управления создать нельзя, но можно создать массив объектов Control. Следующий код создает массив, который содержит элементы управления CommandButton.

```
Private Sub UserForm_Initialize()
   Dim Buttons() As CommandButton
   Cnt = 0
   For Each Ctl In UserForm1.Controls
      If TypeName(Ctl) = "CommandButton" Then
         Cnt = Cnt + 1
         ReDim Preserve Buttons(1 To Cnt)
         Set Buttons(Cnt) = Ctl
      End If
   Next Ctl
End Sub
```

Существует ли разница между сокрытием диалогового окна и его выгрузкой из памяти?

Да. Метод `Hide` оставляет диалоговое окно `UserForm` в памяти, но делает его невидимым. Оператор `Unload` выгружает диалоговое окно, начиная процесс "уничтожения" (вызывая событие `Terminate` объекта `UserForm`) и удаляя диалоговое окно `UserForm` из памяти.

Как обеспечить отображение диалогового окна `UserForm` в момент выполнения пользователем других действий?

По умолчанию каждое диалоговое окно `UserForm` отображается в модальном режиме. Это означает, что перед началом других действий диалоговое окно необходимо закрыть. Можно также создавать немодальные диалоговые окна `UserForm`. Для этого методу `Show` передается аргумент `vbModeless`. Ниже приведен пример такого оператора.

```
UserForm1.Show vbModeless
```

Как отобразить индикатор текущего состояния, пока происходит длительный процесс установки приложения?

Воспользуйтесь диалоговым окном `UserForm`. В главе 15 описано несколько способов создания индикаторов текущего состояния, в том числе постепенное увеличение длины прямоугольника, в течение выполнения процесса.

Как включить фигуры Excel в диалоговое окно `UserForm`?

Фигуры невозможно непосредственно вставить в пользовательские диалоговые окна, но их можно применять косвенно. Начните с создания фигуры на рабочем листе. После этого выделите фигуру и выберите команду Главная⇨Буфер обмена⇨Копировать (Home⇨Clipboard⇨Copy). Активизируйте диалоговое окно `UserForm` и вставьте в него объект `Image`. Нажмите клавишу <F4> для отображения окна Properties (Свойства). Выберите свойство `Picture` и нажмите комбинацию клавиш <Ctrl+V>, чтобы поместить содержимое буфера обмена в элемент управления `Image`. Кроме того, вам может понадобиться присвоить свойству `AutoSize` значение `True`.

Как создать список файлов и папок в диалоговом окне UserForm, чтобы пользователь мог выбрать один из файлов?

Создавать такое диалоговое окно `UserForm` нет необходимости. Для этого существует метод VBA `GetOpenFilename`. Он отображает диалоговое окно, в котором пользователь может выбрать диск, папку и файл.

Мне нужно объединить строки и отобразить их в элементе управления ListBox, но при выполнении этой операции строки отображаются с неправильным выравниванием. Как добиться одинакового выравнивания?

Одним из решений является использование моноширинного шрифта, например Courier New. Но лучше настроить элемент управления `ListBox` на форматирование в два столбца (дополнительная информация приведена в главе 14).

Существует ли простой способ заполнения элементов управления ListBox либо ComboBox значениями с помощью кода VBA?

Да, с помощью массива. Следующий оператор добавляет три значения в элемент управления `ListBox1`:

```
ListBox1.List = Array("Январь", "Февраль", "Март")
```

Существует ли возможность отображения встроенного диалогового окна Excel с помощью кода VBA?

Многие (но не все) диалоговые окна Excel могут быть отображены с помощью метода `Application.Dialogs`. Например, следующий оператор отображает диалоговое окно, которое позволяет форматировать числа в ячейках:

```
Application.Dialogs(xlDialogFormatNumber).Show
```

Этот метод не является вполне надежным, и далеко не все диалоговые окна Excel могут открываться таким образом.

Лучшим методом вызова команд ленты (включая те из них, которые отображают диалоговые окна) является использование метода `ExecuteMso` наряду с указанием имени элемента управления. Ниже приводится пример оператора, в результате выполнения которого отображается диалоговое окно, содержащее параметры форматирования чисел.

```
Application.CommandBars.ExecuteMso("NumberFormatsDialog")
```

Дополнительные сведения по этой теме можно найти в главе 22.

При попытке применить методику, описанную в ответе на предыдущий вопрос, я получил сообщение об ошибке. Почему?

Метод `ExecuteMso` завершится неудачно, если его вызвать в неподходящем контексте. Например, следующий оператор отображает диалоговое окно **Вставить ячейки** (Insert Cells). Но если попытаться вызвать этот оператор в случае выделенной диаграммы или защищенной рабочей книги, появится сообщение об ошибке.

```
Application.CommandBars.ExecuteMso ("CellsInsertDialog")
```

Каждый раз при создании диалогового окна UserForm я добавляю кнопки OK и Отмена. Существует ли способ автоматического добавления этих элементов управления?

Да, такой способ существует. Настройте диалоговое окно `UserForm`, чтобы оно содержало все элементы управления, которые используются наиболее часто. После этого выберите **File⇨Export File** (Файл⇨Экспорт файла) для сохранения этого диалогового окна. Когда возникнет необходимость добавить новое диалоговое окно в проект, выберите команду **File⇨Import File** (Файл⇨Импорт файла).

Существует ли возможность создания диалогового окна `UserForm` без строки заголовка?

Да, но в этом случае понадобятся некоторые сложные API-функции.

Когда я щелкаю на кнопке в диалоговом окне `UserForm`, ничего не происходит. Что я делаю неправильно?

Элементы управления, добавленные в диалоговое окно `UserForm`, не выполняют никакой функции, пока им не будет предоставлена процедура обработки события. Эти процедуры должны располагаться в модуле кода диалогового окна `UserForm`, а также иметь соответствующие имена.

Могу ли я создать диалоговое окно, размер которого будет оставаться постоянным, независимо от текущего разрешения экрана?

Да, такое диалоговое окно можно создать, но стоит ли результат затраченных усилий? Рекомендуется создать код, который определяет разрешение экрана и использует свойство `Zoom` диалогового окна `UserForm` для изменения его размера. Учтите, что все диалоговые окна разрабатываются для разрешения 1024×768, которое наиболее часто применяется в мониторах современных компьютеров.

Существует ли возможность создания диалогового окна `UserForm`, которое позволяет указывать диапазон ячеек с помощью мыши?

Да. Воспользуйтесь элементом управления `RefEdit`. Пример использования этого элемента управления приводится в главе 14.

Существует ли способ изменения расположения диалогового окна `UserForm`?

Существует. Необходимо лишь установить значения свойств `Left` и `Top` диалогового окна `UserForm`. Но чтобы изменение было реализовано, установите свойство `StartUpPosition` в значение `0`.

Я работаю на системе с двумя мониторами, а пользовательские диалоговые окна не отображаются в центре окна Excel. Существует ли метод центрирования пользовательских окон?

Да. Просто воспользуйтесь следующим кодом для отображения пользовательского диалогового окна.

```
With UserForm1
    .StartUpPosition = 0
    .Left = Application.Left + (0.5 * Application.Width) _
       - (0.5 * .Width)
       .Top = Application.Top + (0.5 * Application.Height) _
          - (0.5 * .Height)
    .Show 0
End With
```

Можно ли создать диалоговое окно `UserForm`, размер которого будет изменяться пользователем?

Да. Соответствующий пример рассматривается в главе 15.

Надстройки

Где можно найти надстройки для Excel?

Надстройки для Excel расположены в нескольких местах.

- В Excel интегрировано несколько надстроек, которые можно использовать в любой момент. Для их установки используйте диалоговое окно Надстройки (Add-Ins).
- Дополнительные надстройки можно загрузить с веб-сайта Microsoft Office Update.
- Независимые производители создают надстройки, предназначенные для выполнения самых разных задач.
- Некоторые разработчики создают бесплатные надстройки и распространяют их через веб-сайты.
- Можно создать собственные надстройки.

Как установить надстройку?

Чаще всего надстройки устанавливаются с помощью диалогового окна Надстройки (Add-Ins). Выберите команду Файл⇨Параметры Excel (File⇨Excel Options). В диалоговом окне Параметры Excel выберите вкладку Надстройки (Add-Ins). Затем в списке Управление (Manage) выберите пункт Надстройки Excel (Excel Add-ins) и щелкните на кнопке Перейти (Go). Для более быстрого перехода в диалоговое окно Надстройки (Add-Ins) нажмите комбинацию клавиш <Alt+TI>. Либо, если отображена вкладка ленты Разработчик (Developer), выберите команду Разработчик⇨Надстройки⇨Надстройки (Developer⇨Add-Ins⇨Add-Ins).

Для установки надстройки можно также воспользоваться командой Файл⇨Открыть (File⇨Open), но лучше все же обратиться к диалоговому окну Надстройки (Add-Ins). Если для открытия надстройки применялась команда Файл⇨Открыть (File⇨Open), ее нельзя закрыть без помощи VBA.

При установке собственной надстройки с помощью диалогового окна Надстройки я не обнаружил ее имени и описания. Как добавить описание к надстройке?

Перед созданием надстройки необходимо воспользоваться командой Файл⇨Сведения⇨Свойства⇨Дополнительно (File⇨Info⇨Propertiers⇨Advanced), чтобы открыть диалоговое окно Свойства (Properties). Щелкните на вкладке Все свойства (Summary). В поле Название (Title) введите текст, который будет отображаться в диалоговом окне Надстройки. В поле Примечания (Comments) укажите описание надстройки. Затем создайте надстройку обычным образом.

Некоторые надстройки я больше не использую. Как их удалить из списка в диалоговом окне Надстройки?

Неиспользуемые надстройки невозможно удалить из списка средствами Excel. Один из способов удаления надстроек заключается в удалении файлов, в которых они хранятся. После этого при попытке открыть надстройку из диалогового окна Надстройки (Add-Ins) Excel предложит удалить надстройку из списка. Вам останется согласиться с этим предложением.

Как создать надстройку?

Активизируйте рабочий лист и выполните команду Файл⇨Сохранить как (File⇨Save As). Затем в раскрывающемся списке Сохранить как (Save as Type) выберите пункт Надстройка Excel (Excel Add-in (*.xlam)). После этого создается надстройка, причем исходная рабочая книга остается открытой.

При попытке создать надстройку диалоговое окно сохранения документа не предоставляет возможности выбрать необходимый тип файла

Скорее всего, активный лист не является рабочим листом. Надстройка должна включать хотя бы один рабочий лист, который должен быть активным в момент ее сохранения.

Нужно ли преобразовывать все ключевые рабочие книги в надстройки?

Нет! Несмотря на то что надстройку можно создать на основе любой рабочей книги, не все рабочие книги подходят на эту роль. Когда рабочая книга превращается в надстройку, она становится полностью невидимой. Для большинства рабочих книг невидимость означает невозможность использования.

Существует ли необходимость в хранении двух версий рабочей книги: XLSM и XLAM?

Нет. Надстройку можно изменять, а также легко преобразовывать в рабочую книгу.

Как изменить надстройку после ее создания?

Если нужно изменить только код VBA, никаких дополнительных действий предпринимать не нужно; доступ к коду можно получить в среде VBE, а затем сохранить внесенные в код изменения. Если нужно изменить информацию в рабочем листе, запустите редактор VBE (комбинация клавиш <Alt+F11>) и установите свойство `IsAddIn` объекта `ThisWorkbook` (ЭтаКнига) в значение `False`. Внесите все необходимые изменения, установите свойство `IsAddIn` в значение `True` и повторно сохраните файл.

Какая разница между файлом XLSM и файлом XLAM, который создан на основе файла XLSM? Откомпилирован ли файл XLAM? Может, он работает быстрее?

Между данными файлами не существует особой разницы, и увеличение быстродействия несущественно. Код VBA всегда компилируется перед выполнением. Это касается как файлов XLSM, так и файлов XLAM. Файлы XLAM содержат код VBA, а не откомпилированный код. Также рабочая книга в формате XLAM будет невидимой.

Как защитить код надстройки от просмотра пользователями?

Запустите редактор VBE и выберите команду Tools⇨*xxx* Properties (Сервис⇨Свойства *xxx*) (где *xxx* — название проекта). Перейдите на вкладку Protection (Защита), установите флажок Lock project from viewing (Блокировать просмотр проекта) и введите пароль.

Защищены ли мои надстройки? Другими словами, если я распространяю XLAM-файл, можно ли быть уверенным в том, что больше никто не увидит мой исходный код?

Надстройку можно защитить с помощью пароля. Это предотвратит доступ к исходному коду для большинства пользователей. Новые версии Excel имеют улучшенные возможности по обеспечению безопасности, но остается вероятность взлома пароля с помощью целого ряда утилит. Если это вас не устраивает, то обратитесь к другому способу распространения приложений.

Пользовательский интерфейс

Как с помощью VBA можно добавить на ленту простую кнопку?

Напрямую это невозможно. Можно написать специальный XML-код (известный как код RibbonX) и вставить XML-документ в файл рабочей книги с помощью одной из утилит, любезно предлагаемых независимыми производителями. Либо, если вы являетесь сторонником садомазохизма (и знаете, что собираетесь делать), можете просто разархивировать соответствующий документ и выполнить требуемые правки вручную.

Каким образом можно изменить пользовательский интерфейс, чтобы облегчить пользователям запуск разработанных мною макросов?

В Excel 2010 можно изменить пользовательский интерфейс одним из следующих способов:

- изменить ленту путем добавления кода RibbonX (эта задача довольно сложная);
- добавить макрос на панель быстрого доступа (эта задача выполняется вручную, средствами VBA этого достичь невозможно);
- добавить макрос на ленту (эта задача выполняется вручную, средствами VBA этого достичь невозможно);
- присвоить макросу комбинации клавиш;
- добавить новый элемент в контекстное меню;
- создать "старомодную" панель инструментов или меню, которые будут отображаться на вкладке Надстройки (Add-Ins).

Каким образом можно добавить макрос на панель быстрого доступа?

Щелкните правой кнопкой мыши на панели быстрого доступа, затем в контекстном меню выберите команду Настройка панели быстрого доступа (Customize Quick Access Toolbar). Находясь на вкладке Панель быстрого доступа (Quick Access Toolbar) диалогового окна Параметры Excel (Excel Options), выберите пункт Макросы (Macros) в раскрывающемся списке, который находится в левой части окна. Выберите нужный макрос, затем щелкните на кнопке Добавить (Add). Для изменения отображаемого текста или значка используйте кнопку Изменить (Modify).

Каким образом можно добавить макрос на ленту?

Щелкните правой кнопкой мыши на ленте и в контекстном меню выберите пункт Настройка ленты (Customize the Ribbon). На вкладке Настройка ленты (Customize Ribbon) диалогового окна Параметры Excel (Excel Options) в находящемся слева списке выберите параметр Макросы (Macros). Выберите макрос и щелкните на кнопке Добавить (Add). Обратите внимание на то, что невозможно добавить макрос в существующую группу. Сначала создайте новую группу вкладки путем щелчка на кнопке Создать группу (New Group).

Каким образом с помощью VBA активизировать определенную вкладку ленты?

Для выполнения этой задачи используется метод `SendKeys`. Нажмите клавишу <Alt> для поиска требуемой комбинации клавиш. Например, для выделения вкладки Разметка страницы (Page Layout) воспользуйтесь следующим оператором:

```
Application.SendKeys "%p{F6}"
```

Могу ли я отключить все контекстные меню, вызываемые щелчком правой кнопкой мыши?

Для выполнения этой задачи используется следующая процедура.

```
Sub DisableAllShortcutMenus()
   Dim cb As CommandBar
   For Each cb In CommandBars
      If cb.Type = msoBarTypePopup Then _
         cb.Enabled = False
   Next cb
End Sub
```

Часть VIII

Приложения

В этой части...

Приложение A

Интерактивные ресурсы Excel

В этом приложении...

- Справочная система Excel
- Техническая поддержка со стороны компании Microsoft
- Группы новостей
- Веб-сайты

Если информация, изложенная в этой книге, оказалась вам полезной, то я выполнил свою задачу. Но книга, к сожалению, не является исчерпывающим источником информации. Кроме того, время от времени возникают новые вопросы, поэтому всегда необходимо оставаться в курсе событий.

Таким образом, я систематизировал список ресурсов, которые помогут читателям повысить свою квалификацию как разработчика приложений для Excel. Эти ресурсы разделены на четыре категории: справочная система Excel, техническая поддержка со стороны Microsoft, группы новостей и веб-сайты в Интернете.

Справочная система Excel

Многие пользователи забывают о таком великолепном источнике информации, как справочная система Excel. Для получения доступа к этой системе щелкните на пиктограмме с изображением вопросительного знака в правом верхнем углу окна Excel или нажмите клавишу <F1>. Независимо от выбранного метода появляется новое окно, в котором отображается справка Excel. Затем сформулируйте поисковый запрос и щелкните на кнопке Найти (Search).

Если вы работаете в редакторе Visual Basic, для получения справки воспользуйтесь одним из следующих методов:

- введите в поле, находящееся справа от панели инструментов, поисковый запрос и нажмите клавишу <Enter>;

- переместите мигающий курсор к требуемому ключевому слову, объекту, свойству или методу и нажмите клавишу <F1>.

Справочная система Excel далека от совершенства, поскольку часто предлагает поверхностную справку, полностью игнорируя некоторые вопросы. Но для начинающих пользователей эта система окажет неоценимую поддержку.

Техническая поддержка со стороны компании Microsoft

Техническая поддержка — это распространенный термин. Он означает поддержку производителя программного обеспечения. В данном случае речь идет о поддержке, непосредственно предоставляемой компанией Microsoft. Техническая поддержка Microsoft реализуется в нескольких формах.

Варианты поддержки

Варианты поддержки со стороны компании Microsoft постоянно изменяются. Для просмотра тех из них, которые доступны в настоящий момент времени (на бесплатной и платной основе), обратитесь на сайт Microsoft:

```
http://support.microsoft.com
```

База знаний Microsoft

Наиболее эффективный метод решения проблемы вы найдете в Microsoft Knowledge Base (База знаний Microsoft). Это основной источник информации о продуктах компании Microsoft — обширная база данных, которая поддерживает поиск информации и состоит из десятков тысяч подробных статей, содержащих техническую информацию, а также списков ошибок, исправлений и т.д.

С помощью Интернета можно получить бесплатный и неограниченный доступ к ресурсам базы знаний. Эта база данных расположена по такому адресу:

```
http://support.microsoft.com/search
```

Начальная страница Microsoft Excel

Официальная начальная страница Microsoft Excel находится по такому адресу:

```
http://www.microsoft.com/office/excel
```

Здесь вы найдете массу полезных сведений, включая советы, шаблоны, ответы на вопросы, учебный материал, а также ссылки на продукты компании Microsoft.

Начальная страница Microsoft Office

Для получения сведений о программах Office 2010 (в том числе и Excel) посетите вебсайт Microsoft:

```
http://office.microsoft.com
```

Здесь вы найдете обновления программ, надстройки, примеры, а также много другой полезной информации.

Примечание

Интернет — динамичная, быстро меняющаяся среда. Вследствие этого веб-сайты часто реорганизуются, поэтому некоторые URL-ссылки, приведенные в этом приложении, в какой-то момент могут стать недействительными.

Группы новостей

Служба Usenet — это сеть в пределах Интернета, которая предоставляет доступ к тысячам групп новостей, посвященных определенным темам. В этих конференциях можно пообщаться с людьми, у которых общие интересы.

Существуют тысячи конференций, посвященных темам, которые только можно себе представить (имеются группы новостей, посвященные темам, которые и представить себе нельзя). Обычно на вопросы, размещенные в группах новостей, ответ дается в течение 24 часов (конечно, если вопрос задан таким образом, что у подписчиков возникает желание отвечать).

Доступ к группам новостей с помощью программы чтения новостей

Для обеспечения доступа к группам новостей Usenet используется программа чтения новостей. Существует множество подобных программ, причем одна из них уже установлена на вашем компьютере. В зависимости от используемой версии Windows, эта программа может называться Outlook Express, Почта Windows или Почта Windows Live (загружается и устанавливается отдельно).

Компания Microsoft поддерживает большое количество групп новостей, причем некоторые из них посвящены исключительно Excel. Если ваш провайдер услуг Интернета не поддерживает группы новостей Microsoft, обратитесь непосредственно к серверу новостей Microsoft (на практике этот метод используется чаще всего). Для доступа к серверу новостей Microsoft (находится по адресу `msnews.microsoft.com`) сконфигурируйте соответствующим образом программу чтения новостей (отличную от веб-браузера).

Доступ к группам новостей с помощью браузера

Помимо программы чтения новостей, для просмотра и отправки сообщений в группы новостей Microsoft может использоваться веб-браузер. Конечно, веб-браузер не в состоянии обеспечивать такую скорость, как стандартная программа чтения новостей, поэтому этот способ применяется в том случае, когда доступ к группам новостей запрещен путем выбора соответствующих сетевых политик.

- Получите доступ к тысячам групп новостей на сервере Google Groups по такому адресу:
  `http://groups.google.com`
- Получите доступ к группам новостей Microsoft (включая группы новостей Excel) по такому адресу:
  `www.microsoft.com/communities/newsgroups/default.mspx`

В табл. А.1 перечислены наиболее популярные англоязычные группы новостей Excel, находящиеся на сервере новостей Microsoft (эти же группы новостей доступны на сервере Google Groups).

Таблица А1. Группы новостей, посвященные Excel, на сервере Microsoft

Конференция	Тема
`microsoft.public.excel`	Общие вопросы по работе с Excel
`microsoft.public.excel.charting`	Создание диаграмм Excel
`microsoft.public.excel.interopoledde`	OLE, DDE и другие вопросы взаимодействия приложений
`microsoft.public.excel.macintosh`	Вопросы, связанные с Excel для Macintosh
`microsoft.public.excel.misc`	Общие темы, оказывающиеся за пределами тематических разделов
`microsoft.public.excel.newusers`	Помощь для новичков в Excel
`microsoft.public.excel.printing`	Печать в Excel
`microsoft.public.excel.programming`	Программирование макросов с помощью Excel VBA
`microsoft.public.excel.worksheet.functions`	Шаблоны Spreadsheet Solutions и другие файлы XLT
`microsoft.public.excel.templates`	Функции рабочего листа

Поиск в группах новостей

Наиболее быстрым способом найти ответ на интересующий вас вопрос является поиск среди ответов в группах новостей. В этом случае вы получите готовый результат намного быстрее, чем если бы задавали вопрос в группе новостей. Если ваш вопрос не относится к категории сложных и редко задаваемых, велика вероятность, что кто-то уже спрашивал нечто подобное и даже получил ответ. Лучше всего найти готовый ответ на ваш животрепещущий вопрос — обратиться на сервер Google Groups по адресу `http://groups.google.com`.

Каким образом осуществляется поиск? Предположим, что вас интересует вопрос, связанный с идентификацией уникальных значений в диапазоне ячеек. В этом случае можно выполнить поиск по следующим ключевым словам: **`Excel`**, **`диапазон`** и **`уникальный`**. В результате поиска Google возвращает несколько десятков групп новостей, в которых встречаются эти ключевые слова.

Если количество результатов поиска слишком велико, конкретизируйте поисковый запрос путем добавления дополнительных критериев. Поиск готового ответа займет определенное время, но зато результат оправдает ваши ожидания. Фактически, в Google можно найти ответы на 90% вопросов, задаваемых в группах новостей Excel.

Советы по написанию вопросов в группу новостей

Если вы лишь недавно начали принимать участие в группах новостей, учтите следующие моменты.

1. Удостоверьтесь в том, что на ваш вопрос еще не отвечали. Для этого воспользуйтесь поиском, указав соответствующие ключевые слова.
2. Укажите описательную тему сообщения. Сообщение с темой "Помогите!" или "Еще один вопрос" привлечет к себе намного меньше внимания, чем сообщение с темой "Изменение размера области построения диаграммы".
3. Укажите используемую вами версию Excel. Во многих случаях ответ на заданный вопрос зависит от применяемой версии Excel.

4. В одном сообщении задавайте только один вопрос.
5. По возможности конкретизируйте вопрос.
6. Вопрос должен быть краток. Желательно, чтобы он находился в рамках основной темы. Кроме того, он должен содержать достаточно информации для получения необходимого ответа.
7. Укажите список действий, которые предпринимались для получения решения.
8. Вопрос должен размещаться в подходящей группе новостей, а рассылка сообщений допустима только в том случае, когда сообщение касается нескольких конференций.
9. Не используйте только верхний или нижний регистр; проверьте грамматику и орфографию сообщения.
10. Не вкладывайте в сообщение файлы.
11. Не используйте формат HTML. Для сообщений лучше всего применять формат простого текста.
12. Если нужно получить ответ по электронной почте, не используйте "антиспам" адреса, которые требуют модификации адреса перед отправкой.

    Зачем заставлять выполнять лишнюю работу того, кто и так пытается оказать вам услугу?

Веб-сайты

В Интернете находятся тысячи веб-сайтов, которые посвящены Excel. Приведем несколько самых посещаемых ресурсов.

The Spreadsheet Page

`http://spreasheetpage.com`

Это веб-сайт автора книги. Он содержит файлы, советы разработчикам, инструкции по доступу к скрытым возможностям Excel, а также обширный список ссылок на другие узлы, посвященные электронным таблицам. Помимо этого, на узле приводится информация о книгах автора, здесь вы найдете даже шутки об электронных таблицах.

Блог, посвященный Excel

`http://DailyDoseOfExcel.com`

Часто обновляемый блог, созданный Диком Куслейка. Количество создателей этого блога составляет несколько десятков, среди них — автор этой книги. Здесь рассматривается широкий круг вопросов, причем читатели могут оставлять свои комментарии.

Сайт Йона Пелтиера

`http://peltiertech.com/Excel`

Те, кто часто посещает группу новостей `microsoft.public.excel.charting`, знакомы с Йоном Пелтиером. Йон обладает уникальной способностью решать практически все проблемы, связанные с созданием диаграмм. На его веб-сайте можно найти множество советов по работе с Excel, а также обширную коллекцию диаграмм.

Сайт Чипа Пирсона

`www.cpearson.com/excel.htm`

На этом сайте, поддерживаемом Чипом Пирсоном, можно найти десятки примеров кода VBA, а также ряд примеров применения формул.

Сайт Contextures

`http://contextures.com/`

Этот сайт поддерживается Деборой Далглиш и содержит ряд материалов, посвященных Excel и Access.

Блог Pointy Haired Dilbert

`http://chandoo.org/wp/`

Интересный блог, посвященный Excel.

Сайт Дейвида Макритчи

`www.mvps.org/dmcritchie/excel/excel.htm`

На сайте Дейвида можно найти массу часто обновляемой информации об Excel.

Мистер Excel

`www.MrExcel.com`

Билл Джелен, известный также под именем Мистер Excel, ведет сайт, посвященный Excel. На сайте поддерживается специализированный форум.

Приложение Б

Справочник по операторам и функциям VBA

В этом приложении...

- Вызов функций Excel с помощью операторов VBA

Это приложение содержит полный список всех операторов VBA и встроенных функций. Дополнительная информация приводится в интерактивном справочном руководстве Excel.

Примечание

В Excel 2010 новые операторы VBA не появились.

Таблица Б.1. Операторы VBA

Оператор	Действие
`AppActivate`	Активизирует окно приложения
`Beep`	Выдает звуковой сигнал с помощью встроенного в компьютер динамика
`Call`	Передает управление другой процедуре
`ChDir`	Изменяет текущую папку
`ChDrive`	Изменяет текущий диск
`Close`	Закрывает текстовый файл
`Const`	Объявляет значение константы
`Date`	Устанавливает текущую системную дату
`Declare`	Объявляет ссылку на внешнюю процедуру в библиотеке DLL
`DefBool`	Устанавливает `Boolean` в качестве типа по умолчанию для переменных, имена которых начинаются с определенных символов
`DefByte`	Устанавливает `Byte` в качестве типа по умолчанию для переменных, имена которых начинаются с определенных символов

Продолжение табл. Б.1

Оператор	Действие
DefCur	Устанавливает Currency в качестве типа по умолчанию для переменных, имена которых начинаются с определенных символов
DefDate	Устанавливает Date в качестве типа по умолчанию для переменных, имена которых начинаются с определенных символов
DefDec	Устанавливает Decimal в качестве типа по умолчанию для переменных, имена которых начинаются с определенных символов
DefDbl	Устанавливает Double в качестве типа по умолчанию для переменных, имена которых начинаются с определенных символов
DefInt	Устанавливает Integer в качестве типа по умолчанию для переменных, имена которых начинаются с определенных символов
DefLng	Устанавливает Long в качестве типа по умолчанию для переменных, имена которых начинаются с определенных символов
DefObj	Устанавливает Object в качестве типа по умолчанию для переменных, имена которых начинаются с определенных символов
DefSng	Устанавливает Single в качестве типа по умолчанию для переменных, имена которых начинаются с определенных символов
DefStr	Устанавливает String в качестве типа по умолчанию для переменных, имена которых начинаются с определенных символов
DefVar	Устанавливает Variant в качестве типа по умолчанию для переменных, имена которых начинаются с определенных символов
DeleteSetting	Удаляет параметр или раздел приложения из системного реестра
Dim	Объявляет переменные и (дополнительно) типы данных
Do - Loop	Цикл по набору инструкций
End	Если используется сам по себе, то завершает выполнение программы. Кроме того, этот оператор используется для завершения блока операторов, которые начинаются оператором If, With, Sub, Function, Property, Type или Select
Enum	Объявляет тип перечисления
Erase	Повторно инициализирует массив
Error	Имитирует определенное ошибочное состояние
Event	Объявляет событие, определенное пользователем
Exit Do	Выход из блока операторов Do - Loop
Exit For	Выход из блока операторов Do - For
Exit Function	Выход из процедуры Function
Exit Property	Выход из свойства
Exit Sub	Выход из программы
FileCopy	Копирует файл
For Each - Next	Цикл по набору инструкций (охватывается каждый элемент набора)
For - Next	Цикл по набору инструкций указанное количество раз
Function	Объявляет имя и аргументы процедуры Function
Get	Читает данные из текстового файла
GoSub...Return	Передача управления и возврат из процедуры
GoTo	Переход к оператору в процедуре
If-Then-Else	Выполняет условный оператор

Продолжение табл. Б.1

Оператор	Действие
`Implements`	Указывает интерфейс или класс, которые будут реализованы в модуле класса
`Input #`	Читает данные из текстового файла с последовательным доступом
`Kill`	Удаляет файл с диска
`Let`	Присваивает значение выражения переменной или свойству
`Line Input #`	Читает строку данных из текстового файла с последовательным доступом
`Load`	Загружает, но не отображает объект
`Lock...Unlock`	Управляет доступом к текстовому файлу
`Lset`	Выравнивает строку по правому краю в пределах строковой переменной
`Mid`	Замещает строку символов другими символами
`MkDir`	Создает новую папку
`Name`	Переименовывает файл или папку
`On Error`	Передает управление в случае возникновения ошибки
`On...GoSub`	Выполняет условную передачу управления процедуре
`On...GoTo`	Выполняет условную передачу управления
`Open`	Открывает текстовый файл
`Option Base`	Изменяет нижний предел для массивов, принятый по умолчанию
`Option Compare`	Объявляет режим сравнения, используемый по умолчанию для сравнения строк
`Option Explicit`	Требует объявления всех переменных в модуле
`Option Private`	Указывает, что весь модуль имеет область действия `Private`
`Print #`	Записывает данные в файл с последовательным доступом
`Private`	Объявляет локальный массив или переменную
`Property Get`	Объявляет имя и аргументы процедуры `Property Get`
`Property Let`	Объявляет имя и аргументы процедуры `Property Let`
`Property Set`	Объявляет имя и аргументы процедуры `Property Set`
`Public`	Объявляет общедоступную переменную или массив
`Put`	Записывает переменную в текстовый файл
`RaiseEvent`	Провоцирует возникновение события, определенного пользователем
`Randomize`	Инициализирует генератор случайных чисел
`ReDim`	Изменяет размерность массива
`Rem`	Указывает строку комментария (так же, как и апостроф [''])
`Reset`	Закрывает все открытые текстовые файлы
`Resume`	Продолжает выполнение после завершения процедуры обработки ошибок
`RmDir`	Удаляет пустую папку
`Rset`	Выравнивает строку по правому краю в строковой переменной
`SaveSetting`	Сохраняет или создает параметр приложения в системном реестре
`Seek`	Устанавливает позицию для доступа в текстовом файле
`Select Case`	Выполняет условную обработку операторов
`SendKeys`	Отправляет комбинацию клавиш активному окну
`Set`	Назначает ссылку на объект переменной или свойству
`SetAttr`	Изменяет информацию об атрибутах файла

Окончание табл. Б.1

Оператор	Действие
`Static`	Объявляет переменные на уровне процедуры, причем эти переменные сохраняют значения в процессе выполнения кода
`Stop`	Останавливает выполнение программы
`Sub`	Объявляет имя и аргументы процедуры `Sub`
`Time`	Устанавливает системное время
`Type`	Определяет пользовательский тип
`Unload`	Удаляет объект из памяти
`While...Wend`	Цикл по набору инструкций, выполняемый до тех пор, пока справедливо определенное условие
`Width #`	Устанавливает ширину строки в текстовом файле
`With`	Устанавливает последовательность свойств объекта
`Write #`	Записывает данные в текстовый файл с последовательным доступом

Вызов функций Excel с помощью операторов VBA

Если в VBA не существует функции, эквивалентной функции Excel, то функцию Excel можно вызвать непосредственно в коде VBA. Достаточно перед функцией указать ссылку на объект `WorksheetFunction`. Например, VBA не содержит функции для преобразования радианов в градусы. Так как для этой процедуры в Excel используется отдельная функция, ее можно применять с помощью следующего оператора VBA:

```
Deg = Application.WorksheetFunction.Degrees(3.14)
```

Объект `WorksheetFunction` появился в Excel 97. Для совместимости с более ранними версиями Excel ссылку на объект `WorksheetFunction` можно опустить, а затем записать оператор следующим образом:

```
Deg = Application.Degrees(3.14)
```

Примечание

В Excel 2010 новых функций VBA не появилось.

Таблица Б.2. Функции VBA

Функция	Действие
`Abs`	Возвращает модуль числа
`Array`	Возвращает массив
`Asc`	Преобразует первый символ строки в значение ASCII
`Atn`	Возвращает арктангенс числа
`CallByName`	Выполняет метод или устанавливает (возвращает) свойство объекта
`Cbool`	Приводит выражение к типу `Boolean`
`Cbyte`	Приводит выражение к типу `Byte`
`Ccur`	Приводит выражение к типу `Currency`
`Cdate`	Приводит выражение к типу `Date`

Продолжение табл. Б.2

Функция	Действие
CDbl	Приводит выражение к типу Double
Cdec	Приводит выражение к типу Decimal
Choose	Выбирает и возвращает значение из списка аргументов
Chr	Преобразует код символа в символ
Cint	Приводит выражение к типу Integer
CLng	Приводит выражение к типу Long
Cos	Возвращает косинус числа
CreateObject	Создает объект OLE Automation
CSng	Приводит выражение к типу Single
CStr	Приводит выражение к типу String
CurDir	Возвращает текущую папку
Cvar	Приводит выражение к типу Variant
CVDate	Приводит выражение к типу Date (предназначен для обеспечения совместимости, использовать не рекомендуется)
CVErr	Возвращает определенное пользователем значение ошибки, которое соответствует номеру ошибки
Date	Возвращает текущую системную дату
DateAdd	Добавляет к дате значение времени
DateDiff	Возвращает интервал времени между датами
DatePart	Возвращает указанную часть даты
DateSerial	Преобразует дату в последовательное число
DateValue	Преобразует строку в дату
Day	Возвращает число месяца для определенной даты
DDB	Возвращает амортизацию актива
Dir	Возвращает имя файла или папки, которые соответствуют шаблону
DoEvents	Прекращает выполнение, предоставляя операционной системе возможность обрабатывать другие события
Environ	Возвращает строку с названием рабочей среды
EOF	Возвращает True, если достигнут конец текстового файла
Error	Возвращает число, которое соответствует номеру ошибки
Exp	Возвращает основу натурального логарифма (e), возведенного в степень
FileAttr	Возвращает режим текстового файла
FileDateTime	Возвращает дату и время последней модификации файла
FileLen	Возвращает размер файла в байтах
Filter	Возвращает отфильтрованное подмножество массива символов
Fix	Возвращает целую часть числа
Format	Отображает сообщение в определенном формате
FormatCurrency	Возвращает выражение в формате валюты (определяется настройками системы)
FormatDateTime	Отображает выражение в формате даты и времени
FormatNumber	Отображает выражение в формате числа
FormatPercent	Возвращает процентное выражение
FreeFile	Возвращает следующий доступный номер файла при работе с текстовыми файлами

Продолжение табл. Б.2

Функция	Действие
FV	Возвращает будущее значение ежегодной ренты
GetAllSettings	Возвращает список параметров и разделов системного реестра
GetAttr	Возвращает код, представляющий атрибуты файла
GetObject	Получает из файла объект OLE Automation
GetSetting	Возвращает определенный параметр из раздела приложения системного реестра
Hex	Преобразует десятичное число в шестнадцатеричный формат
Hour	Возвращает час времени
Iif	Оценивает выражение и возвращает одну из двух частей
Input	Возвращает символ из текстового файла с последовательным доступом
InputBox	Отображает окно, запрашивающее данные у пользователя
InStr	Возвращает позицию строки в другой строке
InStrRev	Возвращает позицию строки в другой строке, начиная с конца
Int	Возвращает целую часть числа
Ipmt	Возвращает объем процентных выплат за определенный период ежегодной ренты
IRR	Возвращает внутреннюю ставку прибыли для последовательности денежных потоков
IsArray	Возвращает значение True, если переменная является массивом
IsDate	Возвращает значение True, если переменная является датой
IsEmpty	Возвращает значение True, если переменная не инициализирована
IsError	Возвращает значение True, если выражение является значением ошибки
IsMissing	Возвращает значение True, если необязательный аргумент не передавался процедуре
IsNull	Возвращает значение True, если выражение содержит значение Null
IsNumeric	Возвращает значение True, если выражение рассматривается как число
IsObject	Возвращает значение True, если выражение ссылается на объект OLE Automation
Join	Комбинирует строки, находящиеся в массиве
Lbound	Возвращает наименьшее значение размерности массива
Lcase	Возвращает строку, преобразованную в нижний регистр
Left	Возвращает указанное количество символов строки, начиная слева
Len	Возвращает длину строки в символах
Loc	Возвращает текущую позицию чтения и записи в текстовом файле
LOF	Возвращает длину открытого текстового файла в байтах
Log	Возвращает натуральный логарифм числа
LTrim	Возвращает копию строки без начальных пробелов
Mid	Возвращает указанное количество символов строки
Minute	Возвращает минуту времени
MIRR	Возвращает модифицированную внутреннюю ставку доходности для последовательности периодических денежных потоков
Month	Возвращает месяц даты
MonthName	Возвращает строку, содержащую названия месяцев
MsgBox	Отображает модальное окно сообщения

Продолжение табл. Б.2

Функция	Действие
Now	Возвращает текущие системные время и дату
Nper	Возвращает количество периодов ежегодной ренты
NPV	Возвращает общее текущее значение инвестиций
Oct	Преобразует десятичное значение в восьмеричное
Partition	Возвращает строку, которая представляет диапазон, содержащий значение
Pmt	Возвращает значение выплат для ежегодной ренты
Ppmt	Возвращает значение выплат основной суммы ежегодной ренты
PV	Возвращает текущее значение ежегодной ренты
QBColor	Возвращает код цвета RGB
Rate	Возвращает процентную ставку для периода ежегодной ренты
Replace	Возвращает строку, в которой подстрока замещается другой строкой
RGB	Возвращает число, представляющее значение цвета RGB
Right	Возвращает определенное количество символов, начиная с правого края строки
Rnd	Возвращает случайное число между 0 и 1
Round	Возвращает округленное число
RTrim	Возвращает копию строки без граничных пробелов
Second	Возвращает секунды указанного времени
Seek	Возвращает текущую позицию в текстовом файле
Sgn	Возвращает целое число, которое обозначает знак числа
Shell	Запускает программу
Sin	Возвращает синус указанного числа
SLN	Возвращает прямое обесценивание актива за период времени
Space	Возвращает строку с указанным количеством пробелов
Spc	Позиционирует результат при записи в файл
Split	Возвращает одномерный массив, который содержит заданное количество подстрок
Sqr	Возвращает квадратный корень числа
Str	Возвращает строковое представление числа
StrComp	Возвращает значение, которое указывает результат сравнения строк
StrConv	Возвращает преобразованную строку
String	Возвращает повторяющийся символ или строку
StrReverse	Возвращает строку с обратным порядком символов
Switch	Оценивает список бинарных выражений и возвращает значение, связанное с первым выражением, которое равно True
SYD	Возвращает амортизацию актива за период времени
Tab	Размещает результат при записи в файл
Tan	Возвращает тангенс числа
Time	Возвращает текущее системное время
Timer	Возвращает количество секунд, которые прошли, начиная с полуночи
TimeSerial	Возвращает время для указанного часа, минуты и секунды
TimeValue	Преобразует строку в последовательное значение, представляющее время
Trim	Возвращает строку без начальных и/или завершающих пробелов

Окончание табл. Б.2

Функция	Действие
`TypeName`	Возвращает строку, которая описывает тип данных переменной
`Ubound`	Возвращает наибольшую размерность массива
`Ucase`	Преобразует строку в верхний регистр
`Val`	Возвращает число, которое получено из начальных цифр строки
`VarType`	Возвращает значение, указывающее подтип переменной
`Weekday`	Возвращает число, указывающее день недели
`WeekDayName`	Возвращает строку, содержащую название дня недели
`Year`	Возвращает год указанной даты

Приложение B

Коды ошибок VBA

Это приложение содержит коды всех ошибок, которые можно зафиксировать и обработать. Данная информация пригодится также при обработке ошибок. Для получения дополнительных сведений обратитесь к интерактивному справочному руководству Excel.

Таблица B.1. Коды и описание ошибок VBA

Код ошибки	Описание
3	`Return` без `GoSub`
5	Некорректный вызов процедуры или неправильно заданный аргумент
6	Переполнение (например, значение слишком велико для типа `Integer`)
7	Недостаточно памяти. Данная ошибка редко вызывается недостаточным объемом физической памяти, установленной в системе. Как правило, ошибка указывает на ограниченный объем памяти, который используется Excel или Windows (например, область памяти, которая применяется для хранения изображений или собственных форматов)
9	Элемент за пределами диапазона. Это сообщение об ошибке выводится тогда, когда именованный объект не найден в коллекции объектов. Например, если используется код `Sheets("Лист2")`, а лист `Лист2` не существует
10	Массив имеет постоянные размеры или временно заблокирован
11	Деление на нуль
13	Несоответствие типов
14	Недостаточный размер строки
16	Выражение слишком сложное
17	Невозможно выполнить указанную операцию
18	Возникло прерывание со стороны пользователя. Эта ошибка появляется, если пользователь прерывает выполнение макроса, щелкнув на кнопке Отмена
20	Продолжение работы без ошибки. Это сообщение обычно означает, что перед процедурой обработки ошибки пропущен оператор `Exit Sub`
28	Недостаточный размер стека
35	Подпрограмма или функция не определены
47	Слишком много клиентов библиотеки DLL
48	Ошибка загрузки DLL
49	Неверное соглашение о доступе к DLL
51	Внутренняя ошибка

Продолжение табл. В.1

Код ошибки	Описание
52	Неверное имя или неправильно указано количество файлов
53	Файл не найден
54	Неверный режим файла
55	Файл уже открыт
57	Ошибка ввода/вывода устройства
58	Файл уже существует
59	Неверная длина записи
61	Диск переполнен
62	Достигнут конец файла
63	Неверный номер записи
67	Слишком много файлов
68	Устройство недоступно
70	В доступе отказано
71	Диск не готов
74	Невозможно переименовать диск
75	Ошибка доступа к папке/файлу
76	Папка не найдена
91	Объектная переменная или переменная блока `Witch` не установлена. Эта ошибка возникает в том случае, если при создании объектной переменной не используется метод `Set`. Кроме того, такая ошибка возникает при создании ссылки на объект рабочего листа (например, `ActiveCell`), а в это время активен объект диаграммы
92	Цикл `For` не инициализирован
93	Неверная строка шаблона
94	Неверное использование значения `Null`
96	Невозможно принять событие объекта, так как объект уже отправил события максимальному числу получателей
97	Невозможно вызвать дружественную функцию объекта, который не является экземпляром определяющего класса
98	Свойство или метод не могут содержать ссылку на закрытый объект — ни в виде аргумента, ни в виде возвращаемого значения
321	Неверный формат файла
322	Невозможно создать необходимый временный файл
325	Неверный формат файла ресурса
380	Неверное значение свойства
381	Неверный индекс массива свойств
382	`Set` не поддерживается во время выполнения
383	`Set` не поддерживается (свойство предназначено только для чтения)
385	Необходим индекс массива свойств
387	`Set` не разрешен
393	`Get` не поддерживается во время выполнения
394	`Get` не поддерживается (свойство предназначено только для чтения)
422	Свойство не найдено
423	Свойство или метод не найдены

Окончание табл. В.1

Код ошибки	Описание
424	Необходим объект. Эта ошибка возникает, если текст после точки не распознается как объект
429	Компонент ActiveX не может создавать объекты (зачастую вызвано проблемами регистрации библиотеки, на которую ссылается приложение)
430	Класс не поддерживает автоматизацию или ожидаемый интерфейс
432	Имя файла или имя класса не найдены во время автоматизации
438	Объект не поддерживает это свойство или метод
440	Ошибка средства автоматизации
442	Связь с библиотекой типов или библиотекой объектов для удаленного процесса утеряна. Щелкните на кнопке OK для удаления ссылки
443	Объект `Automation` не содержит значения, принятого по умолчанию
445	Объект не поддерживает это действие
446	Объект не поддерживает именованные аргументы
447	Объект не поддерживает текущие локальные установки
448	Именованный объект не найден
449	Аргумент обязательный
450	Неверное количество аргументов или неверная установка свойства
451	Процедура `Property Let` не определена, а процедура `Property Get` не возвращает объект
452	Неверный порядковый номер
453	Указанная функция DLL не найдена
454	Ресурс кода не найден
455	Ошибка блокировки ресурса кода
457	Этот раздел уже связан с элементом коллекции
458	Переменная имеет тип `Automation`, не поддерживаемый в Visual Basic
459	Объект или класс не поддерживают набор событий
460	Неверный формат содержимого буфера обмена
461	Метод или элемент данных не найдены
462	Удаленный сервер не существует или недоступен
463	Класс не зарегистрирован в локальном компьютере
481	Неверное изображение
482	Ошибка принтера
735	Невозможно сохранить файл в папке `TEMP`
744	Искомый текст не найден
746	Заменяющее значение слишком длинное
1004	Ошибка, определенная приложением или объектом. Довольно распространенное универсальное сообщение об ошибке. Данное сообщение возникает тогда, когда источником ошибки является не VBA. Другими словами, ошибка определяется в Excel (или другом объекте) и передается в VBA. Также эта ситуация возникает в случае, когда ошибка генерируется специально (для этого используется метод `Raise` объекта `Err`), но она не определена в VBA

Приложение Г

Содержимое компакт-диска

В этом приложении...

- Системные требования
- Использование компакт-диска
- Файлы и программы, находящиеся на компакт-диске
- Решение проблем

В этом приложении описывается содержимое компакт-диска, прилагаемого к книге. Изменения содержимого компакт-диска, выполненные в последний момент, описаны в файле `ReadMe`, находящемся в корневой папке компакт-диска.

Информация в этом приложении разбита на следующие четыре категории:

- системные требования;
- использование компакт-диска;
- файлы и программы, находящиеся на компакт-диске;
- решение проблем.

Системные требования

Убедитесь в том, что ваш компьютер удовлетворяет минимальным системным требованиям, перечисленным в этом разделе. Если большинство требований не выполняется, у вас могут возникнуть проблемы с использованием содержимого компакт-диска:

- компьютер с установленной Windows и программой Microsoft Excel 2010;
- устройство чтения компакт-дисков.

Использование компакт-диска

Для установки содержимого компакт-диска на жесткий диск выполните следующие действия.

1. Вставьте компакт-диск в устройство чтения компакт-дисков.

   Примечание. Экран установки не появляется, если свойство автозапуска отключено. В этом случае щелкните на кнопке Пуск (Start) и выберите команду Выполнить (Run). В появившемся диалоговом окне введите `D:\start.exe`. (Букву D замените другой буквой, соответствующей вашему приводу компакт-дисков, — ее вы сможете найти в окне Мой компьютер (My Computer) (или в окне Компьютер, если вы работаете с Windows Vista/Windows 7).) Щелкните на кнопке OK.

2. Появится интерфейс установки содержимого компакт-диска. Он поддерживает простой способ исследования содержимого диска в стиле "указать и щелкнуть".

Файлы и программы, находящиеся на компакт-диске

В следующих разделах вы найдете подробное описание программ и других материалов, представленных на компакт-диске.

Электронная версия книги

На компакт-диске находится полная англоязычная электронная версия книги в формате Adobe Portable Document Format (PDF). Для чтения этих материалов можно воспользоваться программой Adobe Acrobat Reader, которая также находится на компакт-диске.

Файлы примеров

Файлы примеров, рассматриваемые в книге, находятся в отдельных папках с номерами соответствующих глав. За некоторыми исключениями, все эти файлы имеют формат Excel 2007 и следующие расширения:

- `.xlsx` — файл рабочей книги Excel;
- `.xlsm` — файл рабочей книги Excel, содержащий макросы VBA;
- `.xlam` — файл надстройки Excel, содержащий макросы VBA.

В каждой папке с примерами, соответствующей определенной главе (`Chapter3... Chapter30`), находится подпапка `/Rus`, в которой помещены локализованные файлы примеров.

В процессе открытия XLSM-файла Excel может отобразить предупреждение системы безопасности, в котором говорится о том, что макросы отключены. Для включения макросов щелкните на кнопке Параметры (Options) в панели предупреждений системы безопасности и выберите опцию Включить это содержимое (Enable This Content).

Поскольку файлы, находящиеся на прилагаемом к книге компакт-диске, не вызовут каких-либо проблем с безопасностью, скопируйте их в папку на жестком диске и обозначьте эту папку в качестве источника, которому можно доверять. Для этого выполните следующие действия.

1. Откройте окно Проводника (Explorer) и дважды щелкните на значке устройства чтения компакт-дисков, в котором находится прилагаемый к книге компакт-диск.
2. Щелкните правой кнопкой мыши на папке, которая соответствует корневой папке с файлами примеров, и выберите в контекстном меню команду Копировать (Copy).
3. Выберите папку на жестком диске, в которую будут скопированы файлы. Щелкните на ней правой кнопкой мыши и в контекстном меню выберите команду Вставить (Paste).

   Файлы с компакт-диска копируются в подпапку папки, указанной в п. 3.

Для присваивания новой папке статуса доверенного источника выполните следующие действия.

1. Запустите Excel и выберите команду Файл⇨Параметры Excel (File⇨Excel Options) для отображения диалогового окна Параметры Excel (Excel Options).
2. В окне Параметры Excel выберите вкладку Центр управления безопасностью (Trust Center).
3. Щелкните на кнопке Параметры центра управления безопасностью (Trust Center Settings).
4. В диалоговом окне Центр управления безопасностью (Trust Center) выберите вкладку Надежные расположения (Trusted Locations).
5. Щелкните на кнопке Добавить новое расположение (Add New Location) для отображения диалогового окна Надежное расположение Microsoft Office (Microsoft Office Trusted Location).
6. В диалоговом окне Надежное расположение Microsoft Office (Microsoft Office Trusted Location) щелкните на кнопке Обзор (Browse) и найдите папку, где содержатся файлы, скопированные с компакт-диска.
7. Установите флажок Также доверять всем вложенным подпапкам (Subfolders of This Location Are Also Trusted).

После выполнения всех перечисленных действий при открытии XLSM-файлов, находящихся в доверенном расположении, будут активизированы все макросы, причем сообщение системы безопасности не отображается.

Ниже описаны файлы примеров, находящиеся на прилагаемом к книге компакт-диске.

Примечание

Для некоторых глав файлы примеров отсутствуют.

Глава 3

- `array formula examples.xlsx`. Рабочая книга, содержащая различные примеры формул массива.
- `counting and summing examples.xlsx`. Рабочая книга, содержащая примеры формул подсчета и суммирования.
- `megaformula.xlsm`. Рабочая книга, демонстрирующая промежуточные формулы, мегаформулу, а также функцию VBA.

- named formulas.xlsx. Рабочая книга, содержащая ряд примеров именованных формул.
- yearly calendar.xlsx. Рабочая книга, содержащая календарь на год, который был создан с помощью формул массива.

Глава 4

- sample.xlsm. Файл примера, используемый для демонстрации структуры файла рабочей книги.

Глава 6

- worksheet controls.xlsx. Рабочая книга, демонстрирующая использование на рабочем листе элементов управления ActiveX (без макросов).

Глава 7

- comment object.xlsm. Рабочая книга, демонстрирующая некоторые способы управления объектами Comment с помощью VBA.

Глава 8

- timing test.xlsm. Эта рабочая книга демонстрирует преимущество в скорости выполнения, которое достигается в результате объявления для некоторых переменных определенных типов данных.

Глава 9

- sheet sorter.xlsm. Макрос, выполняющий сортировку листов рабочей книги.

Глава 10

- array argument.xlsm. В этой рабочей книге находится пример функции, в качестве аргумента которой используется массив.
- commission functions.xlsm. Эта рабочая книга содержит пример функции, использующей аргумент.
- draw.xlsm. В этой рабочей книге содержится функция, которая случайным образом выбирает ячейки.
- extended date functions.xlsm. Рабочая книга, включающая примеры функций, работающих с датами до 1900 года.
- extended date functions help.docx. Документ Word, описывающий расширенный набор функций по работе с датами.
- key press.xlsm. Рабочая книга, использующая API-функцию для определения, какая клавиша нажата в данный момент времени: <Ctrl>, <Shift> или <Alt>.
- month names.xlsm. Рабочая книга, которая демонстрирует возврат функцией массива.
- mysum function.xlsm. В этой рабочей книге находится функция, которая имитирует функцию Excel СУММ (SUM).
- no argument.xlsm. Рабочая книга, включающая пример функции без аргументов.

- `remove vowels.xlsm`. Рабочая книга, включающая функцию, которая выполняет удаление гласных звуков из аргумента.
- `upper case.xlsm`. Эта рабочая книга включает функцию, выполняющую преобразование текста с применением символов верхнего регистра.
- `win32api.txt`. Текстовый файл, содержащий объявления и константы Windows API.
- `windows directory.xlsm`. Рабочая книга, содержащая API-функцию, применяемую для определения папки Windows.

Глава 11

- `about range selection.xlsm`. Эта рабочая книга включает макрос, который описывает текущий выделенный диапазон.
- `\batch processing`. Папка, содержащая файлы, используемые в процессе пакетной обработки.
- `celltype function.xlsm`. Эта рабочая книга включает функцию, которая описывает тип данных аргумента, находящегося в единственной ячейке.
- `copy multiple selection.xlsm`. В данной рабочей книге находится макрос, который копирует выделенные несмежные диапазоны ячеек.
- `date and time.xlsm`. Эта рабочая книга содержит макрос, отображающий текущие дату и время.
- `delete empty rows.xlsm`. В этой рабочей книге находится макрос, который удаляет все пустые строки в рабочей книге.
- `drive information.xlsm`. Рабочая книга, содержащая макрос, отображающий сведения относительно всех дисков, установленных в системе.
- `duplicate rows.xlsm`. В этой рабочей книге находится макрос, который дублирует строки на основе содержимого ячейки.
- `efficient looping.xlsm`. Данная рабочая книга демонстрирует эффективные способы организации цикла по диапазону ячеек.
- `file association.xlsm`. Эта рабочая книга содержит API-функцию, которая возвращает приложение, связанное с указанным файлом.
- `hide rows and columns.xlsm`. Рабочая книга, включающая макрос, скрывающий все строки и столбцы, не входящие в выделенный диапазон ячеек.
- `inputbox demo.xlsm`. В этой рабочей книге находится макрос, демонстрирующий запрос значения.
- `inrange function.xlsm`. В этой рабочей книге находится функция, которая позволяет определить, содержится ли один диапазон в другом диапазоне.
- `list fonts.xlsm`. В данной рабочей книге находится макрос, который выводит на экран список всех установленных в системе шрифтов.
- `loop vs array fill range.xlsm`. В этой рабочей книге находятся макросы, которые демонстрируют способы заполнения значениями диапазона ячеек.

- `next empty cell.xlsm`. Рабочая книга, включающая макрос, который определяет следующую пустую ячейку в столбце.
- `page count.xlsm`. В этой рабочей книге находится макрос, который подсчитывает количество страниц в рабочей книге, выводимых на печать.
- `printer info.xlsm`. Рабочая книга, содержащая API-функцию, которая возвращает информацию об активном принтере.
- `prompt for a range.xlsm`. В этой рабочей книге находится макрос, который демонстрирует отображение запроса пользователю о выделении диапазона ячеек.
- `range selections.xlsm`. В этой рабочей книге находится макрос, который демонстрирует различные типы выделения диапазона.
- `select by value.xlsm`. Рабочая книга, которая содержит макрос, демонстрирующий выделение ячеек на основе их значений.
- `sorting demo.xlsm`. В этой рабочей книге находится макрос, который демонстрирует способы сортировки массива.
- `\sound`. Папка, содержащая файлы, используемые при демонстрации генерирования звука в Excel.
- `synchronize sheets.xlsm`. Эта рабочая книга содержит макрос, который синхронизирует рабочие листы.
- `\value from a closed workbook`. Папка с файлами, демонстрирующими выборку значений из закрытой рабочей книги.
- `variant transfer.xlsm`. В этой рабочей книге находится макрос, который передает диапазон ячеек в массив данных типа `Variant`.
- `video mode.xlsm`. В данной рабочей книге находится API-функция, которая определяет текущий видеорежим.
- `windows registry.xlsm`. В этой рабочей книге находится макрос, который демонстрирует запись и чтение значений системного реестра.
- `worksheet functions.xlsm`. Данная рабочая книга включает ряд полезных функций рабочего листа, созданных с помощью VBA.

Глава 12

- `data form example.xlsm`. В этой рабочей книге находится макрос, который отображает встроенную форму ввода данных Excel.
- `get directory.xlsm`. Рабочая книга, демонстрирующая два способа запроса у пользователя ввода требуемой папки.
- `inputbox method.xlsm`. В этой рабочей книге находится макрос, который демонстрирует использование метода Excel `InputBox`.
- `prompt for file.xlsm`. Рабочая книга, демонстрирующая запрос имени файла (файлов).
- `ribbon control names.xlsx`. В данной рабочей книге содержатся названия всех элементов управления ленты Excel 2007 и Excel 2010.
- `VBA inputbox.xlsm`. Эта рабочая книга включает макрос, демонстрирующий использование функции VBA `InputBox`.

Глава 13

- `activex worksheet controls.xlsx`. Эта рабочая книга демонстрирует способы использования элементов управления ActiveX на рабочем листе (без макросов).
- `all userform controls.xlsm`. Данная рабочая книга включает пользовательское диалоговое окно `UserForm`, демонстрирующее применение всех доступных элементов управления.
- `get name and sex.xlsm`. Эта рабочая книга включает простой пример пользовательского диалогового окна `UserForm`.
- `newcontrols.pag`. Файл, включающий настроенные элементы управления, которые могут быть импортированы в набор инструментов диалогового окна `UserForm`.
- `spinbutton and textbox.xlsm`. Эта рабочая книга демонстрирует использование двух элементов управления, `SpinButton` и `TextBox`, в пользовательском диалоговом окне `UserForm`.
- `spinbutton events.xlsm`. В этой рабочей книге демонстрируются события элемента управления `SpinButton`.
- `userform events.xlsm`. Эта рабочая книга демонстрирует события пользовательского диалогового окна `UserForm`.

Глава 14

- `change userform size.xlsm`. Эта рабочая книга демонстрирует применение VBA для изменения размеров пользовательского диалогового окна `UserForm`.
- `date and time picker.xlsm`. Эта рабочая книга демонстрирует применение элемента управления `Date and Time Picker`.
- `listbox activate sheet.xlsm`. Данная рабочая книга демонстрирует, как разрешить пользователю выбрать лист с помощью элемента управления `ListBox`.
- `listbox fill.xlsm`. Рабочая книга, демонстрирующая методы заливки элемента управления `ListBox` в пользовательском диалоговом окне.
- `listbox item transfer.xlsm`. Эта рабочая книга показывает передачу опций между двумя элементами управления `ListBox`.
- `listbox move items.xlsm`. Данная рабочая книга позволяет пользователю изменить порядок следования опций в окне элемента управления `ListBox`.
- `listbox multicolumn1.xlsm`. Эта рабочая книга демонстрирует многоколоночный элемент управления `ListBox`, основанный на диапазоне ячеек.
- `listbox multicolumn2.xlsm`. Данная рабочая книга демонстрирует многоколоночный элемент управления `ListBox`, основанный на диапазоне ячеек.
- `listbox multiple lists.xlsm`. Эта рабочая книга применяется для демонстрации отображения нескольких списков с помощью одного элемента управления `ListBox`.
- `listbox select rows.xlsm`. С помощью этой рабочей книги пользователь может выбрать строки рабочего листа (посредством элемента управления `ListBox`).
- `listbox selected items.xlsm`. Эта рабочая книга демонстрирует способы идентификации выделенных элементов в окне элемента управления `ListBox`.

- `listbox unique items1.xlsm`. Данная рабочая книга демонстрирует способы заполнения элемента управления `ListBox` неповторяющимися значениями.
- `listbox unique items2.xlsm`. Вариант примера `listbox unique items1.xlsm`, в котором также происходит сортировка значений.
- `\mediaplayer`. В этой папке находится рабочая книга `mediaplayer.xlsm` (демонстрирует работу элемента управления Media Player), а также примеры аудиофайлов формата MP3.
- `multipage control demo.xlsm`. Эта рабочая книга демонстрирует применение элемента управления `MultiPage` в пользовательском диалоговом окне `UserForm`.
- `queryclose demo.xlsm`. Данная рабочая книга демонстрирует способ предотвращения закрытия пользователем диалогового окна `UserForm` путем щелчка на кнопке Закрыть в строке заголовка.
- `random number generator.xlsm`. Эта рабочая книга демонстрирует пример простой анимации, осуществляемой в окне `UserForm`.
- `range selection demo.xlsm`. Данная рабочая книга демонстрирует применение элемента управления `RefEdit` в пользовательском диалоговом окне `UserForm`.
- `resizable userform api.xlsm`. Рабочая книга, демонстрирующая применение функций Windows API для изменения размеров пользовательского диалогового окна `UserForm`.
- `splash screen.xlsm`. Демонстрация применения диалогового окна `UserForm` в качестве заставки, появляющейся после открытия рабочей книги.
- `userform menus.xlsm`. Данная рабочая книга демонстрирует применение диалогового окна `UserForm` для отображения меню макросов.
- `zoom and scroll sheet.xlsm`. Эта рабочая книга применяется для демонстрации масштабирования и прокрутки рабочего листа в процессе отображения диалогового окна `UserForm`.
- `zoom userform.xlsm`. С помощью этой рабочей книги пользователь может изменить размеры окна `UserForm`.

Глава 15

- `chart in userform.xlsm`. Эта рабочая книга демонстрирует отображение диаграммы в диалоговом окне `UserForm`.
- `\dataform`. В этой папке находится созданная автором книги надстройка Enhanced Data Form.
- `excel lightbox.xlsm`. Эта рабочая книга демонстрирует затемнение окна Excel во время отображения пользовательского диалогового окна `UserForm`.
- `getacolor function.xlsm`. В данной рабочей книге находится функция, которая позволяет пользователю выбрать цвет с помощью элементов управления в окне `UserForm`.

- `modeless userform1.xlsm`. В этой рабочей книге демонстрируется отображение немодального диалогового окна `UserForm`, в котором находятся сведения об активной ячейке.
- `modeless userform2.xlsm`. Более сложная версия рабочей книги `modeless userform1.xlsm`.
- `move controls.xlsm`. С помощью этой рабочей книги пользователь может перемещать элементы управления в окне `UserForm`.
- `msgbox emulation.xlsm`. Рабочая книга, в которой содержится макрос, имитирующий функцию VBA `MsgBox`.
- `multiple buttons.xlsm`. Эта рабочая книга демонстрирует применение модуля класса, с помощью которого единственная процедура может обрабатывать события для нескольких элементов управления в окне `UserForm`.
- `no title bar.xlsm`. В этой рабочей книге используются API-функции для отображения пользовательского диалогового окна `UserForm` без строки заголовка.
- `progress indicator1.xlsm`. Эта рабочая книга применяется для отображения индикатора хода выполнения в окне `UserForm`.
- `progress indicator2.xlsm`. Эта рабочая книга использует элемент управления `MultiPage` для отображения индикатора хода выполнения в пользовательском диалоговом окне `UserForm`.
- `progress indicator3.xlsm`. Эта рабочая книга демонстрирует отображение индикатора хода выполнения в окне `UserForm` путем изменения размеров этого окна.
- `resizable userform.xlsm`. С помощью этой рабочей книги демонстрируется изменение размеров окна `UserForm` пользователем.
- `semitransparent userform.xlsm`. Рабочая книга, демонстрирующая отображение полупрозрачного пользовательского диалогового окна `UserForm`.
- `simulated toolbar.xlsm`. Эта рабочая книга использует пользовательское диалоговое окно `UserForm` для имитации панели инструментов.
- `sliding tile puzzle.xlsm`. Данная рабочая книга содержит пользовательское диалоговое окно `UserForm`, в котором находятся пазлы.
- `splash screen2.xlsm`. Вариант примера `splash screen.xlsm` из главы 14, где пользовательское диалоговое окно `UserForm` лишено строки заголовка.
- `video poker.xlsm`. Рабочая книга, демонстрирующая реализацию игры в видеопокер в пользовательском диалоговом окне `UserForm`.
- `wizard demo.xlsm`. В этой рабочей книге применяется элемент управления `MultiPage` для отображения простого окна `UserForm`, выполняющего функции мастера.

Глава 16

- `simple undo demo.xlsm`. Эта рабочая книга демонстрирует отмену результата выполнения макроса VBA.
- `text tools.xlam`. Надстройка, добавляющая в Excel свойства обработки текста.

- texttools.chm. Файл справки для надстройки text tools.xlam.
- \text tools helpsource. Исходные файлы, используемые для создания файла справки texttools.chm.

Глава 17

- budget pivot table.xlsm. Эта рабочая книга содержит данные, применяемые при создании сводной таблицы.
- normalized data.xlsx. Рабочая книга, демонстрирующая разницу между нормализованными и просуммированными данными.
- reverse pivot table.xlsm. Эта рабочая книга содержит макрос, который преобразует таблицу итогов в таблицу данных, состоящую из трех столбцов.
- simple pivot table.xlsm. Рабочая книга содержит данные, применяемые при создании сводной таблицы.
- survey data pivot tables.xlsm. В этой рабочей книге находится макрос, который генерирует 28 сводных таблиц на основе диапазона данных.

Глава 18

- animated charts.xlsm. Рабочая книга, демонстрирующая применение VBA для создания анимированных диаграмм.
- chart active cell.xlsm. Эта рабочая книга включает макрос, который отображает диаграмму, использующую данные, основанные на положении активной ячейки.
- chart image map.xlsm. Эта рабочая книга использует события диаграммы для создания простой карты изображения.
- chart in userform.xlsm. Эта рабочая книга отображает диаграмму в окне UserForm. При этом используются данные, основанные на положении активной ячейки.
- climate data.xlsx. Интерактивная диаграмма, не использующая макросы.
- data labels.xlsm. Рабочая книга содержит макрос, который применяет данные о подписях диаграммы, сохраненных в диапазоне.
- events - chart sheet.xlsm. Рабочая книга, которая демонстрирует события для диаграммы, находящейся на листе диаграммы.
- events - embedded chart.xlsm. Рабочая книга, демонстрирующая события для внедренной диаграммы.
- export all graphics.xlsm. Рабочая книга, которая содержит макрос, экспортирующий все графические объекты в рабочей книге.
- format all charts.xlsm. В этой рабочей книге находится макрос, который изменяет форматирование всех диаграмм, находящихся на рабочем листе.
- get series ranges.xlsm. Эта рабочая книга содержит функции, идентифицирующие используемые диаграммой диапазоны.
- hide and unhide series.xlsm. Эта рабочая книга отображает флажки, с помощью которых пользователь выбирает ряды данных, отображаемые на экране.

- hypocycloid - animated.xlsm. Эта рабочая книга включает макрос, применяемый для отображения анимированной диаграммы в форме гипоциклоиды.
- mouseover event - chart sheet.xlsm. Рабочая книга демонстрирует применение события MouseOver для листа диаграммы.
- mouseover event - embedded.xlsm. Эта рабочая книга демонстрирует событие MouseOver для внедренной диаграммы.
- scrolling chart.xlsm. Данная рабочая книга применяется для демонстрации приемов, используемых при создании анимированной прокручиваемой диаграммы.
- size and align charts.xlsm. В состав этой рабочей книги входит макрос, который изменяет размеры, а также выравнивает все диаграммы на рабочем листе.
- sparkline report.xlsm. Рабочая книга, генерирующая отчет, который описывает все спарклайны на рабочем листе.
- unlinked chart.xlsm. Эта рабочая книга включает макрос, который демонстрирует два способа разрыва связи между диаграммой и ее исходными данными.
- vba clock chart.xlsm. В состав этой рабочей книги входит диаграмма, имитирующая обычные стрелочные часы.

Глава 19

- application event tracker.xlsm. С помощью этой рабочей книги демонстрируется контроль событий уровня приложения.
- hide columns before printing.xlsm. Эта рабочая книга использует событие для сокрытия столбцов перед выполнением печати и отображением их после завершения печати.
- log workbook open.xlsm. Данная рабочая книга демонстрирует, каким образом отслеживается каждая рабочая книга, которая была открыта с помощью модуля класса.
- make formulas bold.xlsm. Эта рабочая книга демонстрирует использование события Worksheet Change.
- no shortcut menus.xlsm. В этой рабочей книге используется событие Workbook_Open для отключения комбинаций клавиш, а также событие Workbook_BeforeClose для повторного включения комбинаций клавиш.
- onkey event demo.xlsm. Эта рабочая книга демонстрирует применение события OnKey.
- ontime event demo.xlsm. Данная рабочая книга демонстрирует использование события OnTime.
- shade active row and column.xlsm. Рабочая книга, которая использует событие Worksheet SelectionChange для применения закрашивания к строке и столбцу активной ячейки.
- validate entry1.xlsm. Эта рабочая книга демонстрирует процесс проверки данных, вводимых в ячейку, с помощью VBA (при этом используется свойство EnableEvents).

- validate entry2.xlsm. Эта рабочая книга демонстрирует проверку вводимых в ячейку данных, осуществляемую с помощью VBA (при этом используется статическая переменная).
- validate entry3.xlsm. В этой рабочей книге демонстрируется проверка данных с помощью свойства проверки данных Excel. При этом также гарантируется неизменность условий проверки данных.
- workbook_beforeclose workaround.xlsm. Эта рабочая книга демонстрирует способ устранения проблемы, связанной с использованием события Workbook BeforeClose.

Глава 20

- \automate excel. Папка, в которой находится документ Word, содержащий макрос, который демонстрирует автоматизацию в Excel.
- control panel dialogs.xlsm. Эта рабочая книга содержит макрос, который отображает диалоговые окна панели управления Windows.
- make memos.xlsm. Рабочая книга, которая реализует автоматизацию в Word, а также демонстрирует создание пользовательской заметки.
- personalized email - outlook.xlsm. Эта рабочая книга включает макрос, который отсылает электронное сообщение с помощью Outlook (используется раннее связывание).
- personalized email - outlook (late binding).xlsm. В этой рабочей книге находится макрос, который отсылает электронное сообщение с помощью Outlook (используется позднее связывание).
- personalized email - sendkeys.xlsm. Рабочая книга, включающая макрос, который отсылает электронное сообщение с помощью приложения Почта Windows.
- send pdf via outlook.xlsm. Эта рабочая книга содержит макрос, отсылающий электронное сообщение, к которому прикреплен PDF-файл, с помощью Outlook.
- \shellexecute. Папка, содержащая рабочую книгу, в которой находится макрос, который демонстрирует использование API-функции ShellExecute. В этой папке находится файл shellexecute examles.xlsm и ряд вспомогательных файлов.
- start calculator.xlsm. Эта рабочая книга содержит макрос, который запускает приложение калькулятора.

Глава 21

- check addin.xlam. Рабочая книга, которая содержит код, гарантирующий правильную установку надстройки.
- export charts.xlsm. Рабочая книга Export Charts Utility, которая может быть преобразована в надстройку.
- export charts.chm. Файл справки для рабочей книги export charts.xlsm.
- \export charts help source. Папка с исходными файлами, используемыми для создания файла справки export charts.chm.

- `list add-in information.xlsm`. Рабочая книга включает макрос, отображающий сведения относительно всех надстроек.

Глава 22

- `dynamicmenu.xlsm`. Эта рабочая книга демонстрирует использование элемента управления `DynamicMenu`.
- `mso image browser.xlsm`. Эта рабочая книга содержит макрос, который отображает изображения, связанные с командами ленты.
- `old-style toolbar.xlsm`. Эта рабочая книга демонстрирует создание панели управления, используемой в предыдущих версиях Excel.
- `page break display add-in.xlam`. Надстройка, которая добавляет полезный элемент управления на ленту Excel.
- `page break display.xlsm`. Этот файл рабочей книги применяется для создания надстройки `page break display add-in.xlam`.
- `ribbon control names.xlsx`. Рабочая книга, включающая имена всех элементов управления ленты Excel 2007 и Excel 2010.
- `ribbon controls demo.xlsm`. Рабочая книга, демонстрирующая несколько типов элементов управления ленты.
- `ribbon modification.xlsm`. Эта рабочая книга включает простой пример, который изменяет ленту Excel.

Глава 23

- `add to cell shortcut.xlsm`. Рабочая книга, содержащая макрос, который добавляет новый элемент в контекстное меню.
- `context-sensitive shortcut menu.xlsm`. Рабочая книга с макросом, создающим новое контекстно-зависимое меню.
- `make xl 2003 menus.xlsm`. Рабочая книга с макросом, который добавляет панель инструментов в стиле меню Excel 2003.
- `shortcut with submenu.xlsm`. Рабочая книга, которая содержит макрос, добавляющий в контекстное меню новые элементы меню и подменю.
- `show faceids.xlsm`. Эта рабочая книга включает макрос, который отображает изображения `FaceId`.
- `show shortcut menu items.xlsm`. Эта рабочая книга содержит макрос, который отображает все элементы контекстного меню.
- `show shortcut menu names.xlsm`. Данная рабочая книга включает макрос, который отображает названия всех контекстных меню.

Глава 24

- `\cell comments`. В этой папке находится рабочая книга, которая демонстрирует использование комментариев к ячейке для отображения справки.
- `\function help`. Эта рабочая книга демонстрирует отображение справки для пользовательских функций рабочего листа VBA.

- \html help. В этой папке находятся файлы, которые демонстрируют использование компилированной справки HTML.
- \mhtml file. В этой папке находятся файлы, которые демонстрируют применение MHTML-файла для отображения справки в окне Internet Explorer.
- \textbox. В этой папке содержится рабочая книга, которая демонстрирует использование текстового поля для отображения справки.
- \userform1. В этой папке находится рабочая книга, которая демонстрирует использование пользовательского диалогового окна UserForm с элементом управления SpinButton, применяемым для отображения справки.
- \userform2. В этой папке находится рабочая книга, демонстрирующая использование окна UserForm, в котором находится прокручиваемый элемент управления Label, предназначенный для отображения справки.
- \userform3. В этой папке находится рабочая книга, демонстрирующая использование пользовательского диалогового окна UserForm, в котором находится элемент управления ComboBox, отображающий справку.
- \web browser. В этой папке находятся файлы, которые демонстрируют использование окна UserForm для отображения справки.
- \worksheet. В этой папке находится файл, демонстрирующий использование рабочего листа для отображения справки.

Глава 25

- loan amortization wizard.xlam. Эта надстройка представляет собой мастер расчета займа.

Глава 26

- multilingual wizard.xlsm. Эта рабочая книга представляет собой мастер, в окне которого выбираются различные языки.

Глава 27

- create file list.xlsm. Эта рабочая книга включает макрос, который создает список всех файлов, находящихся в папке.
- export and import csv.xlsm. Данная рабочая книга включает макрос, который экспортирует и импортирует файл CSV.
- export to HTML.xlsm. Эта рабочая книга содержит макрос, который экспортирует данные рабочего листа в HTML-файл.
- export to XML.xlsm. Рабочая книга, содержащая макрос, который экспортирует данные рабочего листа в XML-файл.
- file functions.xlsm. Рабочая книга, включающая функции FileExists и PathExists.
- file information.xlsm. Макрос этой рабочей книги создает список файлов, содержащий также расширенную информацию о файлах.
- \filter text file. В этой папке находятся файлы, используемые для импорта выбранной в текстовом файле информации.

- `recursive file list.xlsm`. Рабочая книга содержит макрос, который создает список файлов, содержащихся в папке, включая все подпапки.
- `show drive info.xlsm`. В этой рабочей книге содержится макрос, который отображает информацию обо всех дисках, установленных в системе.
- `\simple ADO 1`. В этой папке находится пример использования техники ADO для запроса данных в файле Access.
- `\simple ADO 2`. В этой папке находится пример использования техники ADO для запроса данных в текстовом файле CSV.
- `unzip a file.xlsm`. Эта рабочая книга включает макрос, который разархивирует файл.
- `zip files.xlsm`. Эта рабочая книга включает макрос, который архивирует файл.

Глава 28

- `add 100 buttons.xlsm`. В этой рабочей книге находится макрос, который добавляет 100 элементов управления `CommandButton` и код в пользовательское диалоговое окно `UserForm` во время разработки.
- `add button and code.xlsm`. В данной рабочей книге находится макрос, который добавляет кнопку на рабочий лист, а также код VBA, который вызывается после щелчка на кнопке.
- `create userform on the fly.xlsm`. Рабочая книга, содержащая макрос, который создает диалоговое окно `UserForm`.
- `getoption function.xlsm`. Рабочая книга, включающая функцию, которая создает пользовательское диалоговое окно `UserForm` (с элементами управления `OptionButton`) в быстром режиме, а также возвращает значение, которое соответствует выбору пользователя.
- `list all procedures.xlsm`. Рабочая книга содержит макрос, выводящий список всех процедур VBA, находящихся в рабочей книге.
- `list VB components.xlsm`. В этой рабочей книге находится макрос, который выводит список всех компонентов VB, находящихся в рабочей книге.
- `\updateUserBook`. Папка, включающая рабочую книгу с макросом, который заменяет модуль VBA новым модулем.

Глава 29

- `csv class.xlsm`. Рабочая книга, облегчающая процесс импорта/экспорта файла CSV.
- `keyboard class.xlsm`. Рабочая книга, которая содержит модуль класса, определяющий классы `NumLock`, `CapsLock` и `ScrollLock`.

Глава 30

- `chart colors.xlsm`. Рабочая книга содержит макрос, предназначенный для работы с цветами диаграммы.
- `chart to grayscale picture.xlsm`. В этой рабочей книге находится макрос, который создает на основе диаграммы изображение в градациях серого.

- `color conversion functions.xlsm`. В этой рабочей книге находятся функции, которые выполняют преобразования между различными цветовыми системами.
- `document theme demo.xlsx`. Рабочая книга, содержащая различные элементы, которые демонстрируют результаты применения разнообразных тем.
- `generate theme colors.xlsm`. В этой рабочей книге находится макрос, который демонстрирует цвета темы.
- `rgb color demo.xlsm`. Рабочая книга, содержащая интерактивную демонстрацию цветовой системы RGB.
- `\shape object colors`. В этой папке находится рабочая книга с макросом, который предназначен для работы с фигурами.
- `tintandshade demo.xlsm`. Эта рабочая книга демонстрирует работу свойства `TintAndShade`.

Решение проблем

Если у вас возникли проблемы в процессе установки компакт-диска либо использования его содержимого, попытайтесь устранить их одним из следующих способов.

- *Отключите выполняемые на вашем компьютере антивирусные программы.* Устанавливаемые программы зачастую похожи на компьютерные вирусы, поэтому антивирусные программы начинают блокировать мнимую вирусную атаку. (После завершения установки не забудьте снова включить антивирусную защиту.)
- *Завершите все выполняемые программы.* Чем больше выполняется программ, тем меньше доступный объем оперативной памяти. Программы установки также обновляют файлы и программы; поэтому, если в момент установки запущены другие программы, установка может завершиться неудачей.
- *Прочтите файл ReadMe.* Прочтите файл ReadMe, находящийся в корневой папке компакт-диска, для получения новейших сведений относительно устанавливаемых программ.

Предметный указатель

И

К

Л

М

Н

О

П

Р

С

Т

У

Ф

Ц

Ш

Э

Я

Научитесь профессионально работать с Excel и VBA

Изучите возможности языка VBA (Visual Basic for Applications), и вы получите в свое распоряжение практически безграничные ресурсы Microsoft Excel 2010. Джон Уокенбах, один из крупнейших специалистов в области электронных таблиц, щедро поделится с вами своими знаниями и поможет профессиональными советами, которыми буквально переполнена эта великолепная книга.

Даже для опытных пользователей Excel книга станет бесценным источником знаний. Вы ознакомитесь с инновационным подходом к изучению Excel и с этапами разработки приложений электронных таблиц. Вы научитесь разрабатывать процедуры и функции VBA, изучите передовые методики программирования и освоите ряд приемов, знание которых сделает вас подлинным гуру в области VBA-программирования. Если же вы осуществляете переход с одной из предыдущих версий Excel, то благодаря книге сможете легко и быстро освоить новые возможности Excel 2010.

Джон Уокенбах по праву считается одним из лучших специалистов по Excel. Он написал сотни статей и разработал получивший множество наград пакет утилит Power Utility Pak. Из-под его пера вышло более пятидесяти книг, включая такие бестселлеры, как *Excel 2010. Библия пользователя* и *Формулы в Microsoft Excel 2010*. Посетите его персональный сайт *www.spreadsheetpage.com*.

Основные темы книги

- Создание мощных приложений Excel, использующих возможности VBA
- Разработка дружественных диалоговых окон
- Расширение возможностей Excel с помощью пользовательских функций рабочего листа
- Создание VBA-кода, управляемого событиями

На компакт-диске:
- файлы примеров для всех упражнений книги
- электронная англоязычная версия книги

Подробности приведены в приложении Г.

Предмет рассмотрения: Microsoft Excel 2010/VBA
Уровень: для пользователей средней и высокой квалификации

Посетите сайт автора по адресу:
www.spreadsheetpage.com

ДИАЛЕКТИКА
www.dialektika.com

WILEY
www.wiley.com

ISBN 978-5-8459-1721-8

9 785845 917218 11033