Отличия от BASIC

2

В ЭТОЙ ГЛАВЕ

Этот непонятный код

Код VBA способен смутить каждого, кто изучал в школе один из процедурных языков программирования наподобие BASIC или COBOL. Несмотря на то что VBA расшифровывается как "Visual Basic for Applications", он представляет собой *объектно-ориентированную* версию BASIC. Рассмотрим небольшой фрагмент кода VBA.

```
Selection.End(xlDown).Select
Range("A14").Select
ActiveCell.FormulaR1C1 =
"'Всего"
Range("E14").Select
Selection.FormulaR1C1 =
"=SUM(R[-12]C:R[-1]C)"
Selection.AutoFill
Destination:=Range("E14:G14"), _
Type:=xlFillDefault
```

Бьюсь об заклад, что этот код не будет иметь ни малейшего смысла для тех, кто изучал только процедурные языки программирования (к сожалению, если вы старше 25 лет, то в школе вы начинали именно с процедурного языка).

Ниже приведен фрагмент кода, написанный на языке BASIC.

```
For x = 1 to 10
   Print Rpt$(" ", x);
   Print "*";
Next x
```

В результате его выполнения на экране компьютера появится "лесенка" из символов звездочки.

```
*
 *
  *
   *
    *
     *
      *
       *
        *
         *
          *
```

Если вы когда-либо изучали какой-либо процедурный язык программирования, то, глядя на этот код, сразу же поймете, что происходит. Синтаксис процедурного языка программирования больше похож на синтаксис английского языка, нежели синтаксис объектно-ориентированного языка программирования. К примеру, выражение `Print "Привет"` записано в привычном формате "глагол–объект". А теперь постараемся забыть о программировании и рассмотрим один конкретный пример.

Учимся понимать "речь" VBA

Попробуем сыграть в футбол на языке BASIC. Команда "ударить по мячу" на английском будет выглядеть примерно так:

```
Kick the Ball
```

Именно так мы и говорим в повседневной жизни. Глагол "ударить" (kick) следует перед существительным "мяч" (the ball). Аналогично, в приведенном выше примере глагол `Print` следует перед существительным `*` (звездочка).

К сожалению, подобный синтаксис не употребляется ни в одном объектно-ориентированном языке, включая VBA. Исходя из самого названия этого класса языков программирования, становится ясно, что центральное место здесь отводится объекту, т.е. существительному. Команда "ударить по мячу", записанная на языке VBA, будет выглядеть так:

```
Ball.Kick
```

В VBA существительное (*объект*) записывается перед глаголом (*методом*). Базовая структура большинства строк VBA выглядит так:

```
Объект.Метод
```

К сожалению, это не очень похоже на повседневную речь. Никто не говорит "Вода.Пить", "Мяч.Ударить" или "Девушка.Целовать". Именно поэтому VBA кажется очень сложным по сравнению с процедурными языками программирования.

Продолжим аналогию. Представьте, что вы стоите на зеленом газоне перед тремя мячами: футбольным, баскетбольным и бейсбольным. Как сказать на языке VBA "ударить футбольный мяч" члену школьной футбольной команды?

Выше была приведена команда "ударить по мячу" (`Ball.Kick`), однако в данном случае вы не уверены, по какому мячу будет выполнен удар. Скорее

всего, ребенок ударит мяч, который находится ближе к нему, чем остальные (например, бейсбольный).

В VBA практически для каждого объекта (существительного) определяется *коллекция* этих объектов. Рассмотрим электронную таблицу Excel. Строке соответствует набор строк, столбцу — набор столбцов, рабочему листу — набор рабочих листов. С точки зрения синтаксиса имя коллекции объектов составляется из имени объекта и суффикса "s", например:

```
Row⇨Rows,
Cell⇨Cells,
Ball⇨Balls.
```

При обращении к элементу коллекции следует точно указать, какой именно элемент имеется в виду. Существует несколько способов обращения к элементу коллекции. Первый из них состоит в использовании порядкового номера элемента, например:

```
Balls(2).Kick
```

Несмотря на то, что приведенная выше запись вполне корректна, переупорядочение со временем мячей в коллекции может привести к весьма плачевному результату.

Второй способ обращения к элементу коллекции более безопасен и состоит в использовании имени элемента, например:

```
Balls("Soccer").Kick
```

Теперь можно быть уверенным, что ребенок ударит именно по футбольному мячу. Для большинства методов (глаголов) в Excel VBA определены параметры, характеризующие способ выполнения метода (назовем их наречиями). Ниже приведена команда "сильно ударить футбольный мяч так, чтобы он полетел влево":

```
Balls("Soccer").Kick Direction:=Left, Force:=Hard
```

Комбинации двоеточия и знака равенства в коде VBA всегда указывают на параметр метода, и, глядя на эти параметры, вы всегда можете сказать, как будет выполняться действие.

Методы могут иметь много параметров, как обязательных, так и нет. Предположим, что у метода `Kick` есть параметр `Elevation` ("поднятие"). Ниже приведена команда "сильно ударить футбольный мяч так, чтобы он полетел высоко влево":

```
Balls("Soccer").Kick Direction:=Left, Force:=Hard, Elevation:=High
```

Здесь мы подходим к крайне удивительному моменту. Для каждого метода существует определенный порядок следования его параметров, заданный по умолчанию. Если вы не педантичный программист и знаете порядок параметров, то можете опустить их имена, указывая только их значения. Следующая строка кода полностью эквивалентна предыдущей:

```
Balls("Soccer").Kick Left, Hard, High
```

Практика пропуска имен параметров не вносит ясности в код, так как, не зная точного порядка следования параметров, сложно судить о назначении

того или иного значения. Значения параметров `Left`, `Hard` и `High` сами по себе информативны, однако так бывает далеко не всегда. Рассмотрим следующую строку кода:

```
WordArt.Add Left:=10, Top:=20, Width:=100, Height:=200
```

Если пропустить имена параметров, она будет выглядеть так:

```
WordArt.Add 10, 20, 100, 200
```

Несмотря на то что приведенная выше строка кода вполне корректна, отсутствие имен параметров серьезно затрудняет восприятие ее смысла. Точный порядок следования параметров метода можно узнать, обратившись к разделу справочной системы, посвященному этому методу.

Ситуацию усложняет еще и то, что имена параметров обязательно указывать только в случае нарушения стандартного порядка их следования. Ниже приведены две эквивалентные строки кода, соответствующие команде "ударить футбольный мяч так, чтобы он полетел высоко влево" (неважно, насколько сильным будет сам удар):

```
Balls("Soccer").Kick Direction:=Left, Elevation:=High
Balls("Soccer").Kick Left, Elevation:=High
```

Указав имя одного параметра, следует указать также имена всех параметров, которые последуют за ним в этой строке кода.

Некоторые методы не имеют параметров. Ниже приведен код, имитирующий нажатие клавиши <F9>:

```
Application.Calculate
```

Другие методы выполняют действие и возвращают его результат. Ниже приведен код, добавляющий рабочий лист:

```
Worksheet.Add Before:=Worksheets(1)
```

Поскольку метод `Worksheet.Add` создает новый объект, результат его выполнения может быть присвоен переменной (параметры метода при этом следует взять в скобки):

```
Set MyWorksheet = Worksheet.Add (Before:=Worksheets(1))
```

Напоследок рассмотрим еще один важный элемент грамматики — прилагательные. Прилагательные описывают существительные; в Excel аналогичную роль исполняют свойства объектов.

Обратимся к примеру. В Excel существует объект, соответствующий активной ячейке, — `ActiveCell`. Предположим, что нам необходимо изменить цвет активной ячейки на желтый. Цвет ячейки определяется значением свойства `Interior.ColorIndex` объекта `ActiveCell`. Изменение цвета ячейки на желтый описывается следующей строкой кода:

```
ActiveCell.Interior.ColorIndex = 6
```

Обратите внимание, что в приведенном выше коде используется конструкция *`Объект.Свойство`*, похожая на уже рассмотренную нами конструкцию *`Объект.Метод`*. На первый взгляд, их нельзя отличить. Если же присмотреться повнимательнее, то можно заметить отсутствие двоеточия перед знаком ра-

венства в строке с конструкцией *Объект.Свойство*. Обычно свойство всегда присутствует в левой или правой части выражений, связанных с присвоением значения.

Ниже приведена команда, изменяющая цвет текущей ячейки на цвет ячейки A1:

```
ActiveCell.Interior.ColorIndex = Range("A1").Interior.ColorIndex
```

Здесь `Interior.ColorIndex` — это свойство. Изменяя его значение, мы изменяем цвет ячейки. Сравнивая свойство с прилагательным, получаем достаточно странный результат — изменение прилагательного влечет за собой выполнение действия в ячейке. Там, где человек говорит: "Окрась ячейку в желтый цвет", VBA говорит:

```
ActiveCell.Interior.ColorIndex = 30
```

В табл. 2.1 перечислены "части речи" VBA.

Таблица 2.1. Словарь терминов VBA

Термин VBA	Аналог	Примечания
Объект	Существительное	—
Коллекция	Существительное во множественном числе	Обычно указывается элемент коллекции, например `Worksheets(1)`
Метод	Глагол	*Объект.Метод*
Параметр	Наречие	Параметры указываются после имени метода. Между именем параметра и его значением ставится двоеточие и знак равенства (`:=`)
Свойство	Прилагательное	Обычно свойство присутствует в левой или правой части выражения, связанного с присвоением значения, например `ActiveCell.Height = 10` или `x = ActiveCell.Height`

Действительно ли VBA сложен? Нет!

Знание того, с чем вы оперируете, — со свойством или методом, поможет вам использовать в коде корректный синтаксис. Не волнуйтесь, если вы все еще не научились отличать метод от свойства. Если вы создаете код VBA "с нуля", вы должны знать, что изменяет цвет ячейки — глагол или прилагательное, т.е. метод или свойство.

Именно здесь вам пригодится средство записи макросов. Чтобы узнать, как запрограммировать то или иное действие, запишите его в виде макроса и затем изучите сгенерированный код.

Справочная система VBA — нажмите <F1>

Это великолепный инструмент, однако, прежде вам придется пройти одно испытание. Приступая к написанию макросов, *обязательно* убедитесь в наличии на вашем компьютере справочной системы VBA. К сожалению, она не входит в стандартную установку Microsoft Office. Чтобы проверить наличие справочной системы VBA, выполните следующие действия.

1. Запустите Excel и откройте окно редактора Visual Basic, воспользовавшись комбинацией клавиш <Alt+F11>. Выберите команду меню Insert⇨Module (рис. 2.1).
2. Введите три строки кода, как показано на рис. 2.2, и установите курсор посредине слова `MsgBox`.

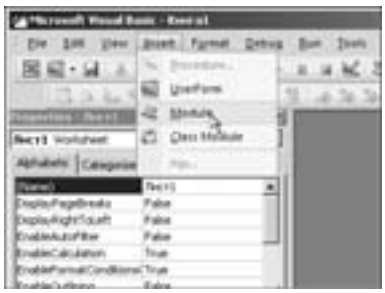


Рис. 2.1. Вставьте в рабочую книгу новый модуль

Рис. 2.2. Установите курсор посредине слова `MsgBox` и нажмите клавишу <F1>

3. Нажмите клавишу <F1>. Если справочная система VBA установлена, откроется окно, показанное на рис. 2.3.

Если справочная система VBA не установлена, Excel выдаст сообщение об ошибке. Установите справочную систему VBA, воспользовавшись компакт-дисками Microsoft Office (при необходимости обратитесь за помощью к системному администратору). Во время инсталляции задайте *выборочную* установку и отметьте файлы справки VBA.

Просмотр разделов справочной системы

Раздел справочной системы, посвященный тому или иному методу или функции, содержит подробное описание всех соответствующих аргументов. В нижней части окна, как правило, приводится пример использования данного метода (рис. 2.4).

Рис. 2.3. Если справочная система VBA установлена, вы увидите на экране это окно

Рис. 2.4. Многие разделы справочной системы VBA содержат пример

Код примера можно выделить (рис. 2.5), скопировать в буфер обмена с помощью комбинации клавиш <Ctrl+C>, а затем вставить в модуль с помощью комбинации клавиш <Ctrl+V>.

Рис. 2.5. Выделите код примера и скопируйте его в буфер обмена с помощью комбинации клавиш <Ctrl+C>

Код записанных макросов наверняка содержит много незнакомых объектов и методов. Установите курсор посредине интересующего вас ключевого слова и нажмите клавишу <F1>, чтобы отобразить соответствующий раздел справочной системы VBA.

Изучение кода записанного макроса

Рассмотрим код первого макроса, записанного в главе 1, и попытаемся понять его смысл в контексте объектов, свойств и методов. Также мы посмотрим, можно ли исправить ошибки, допущенные средством записи макросов.

Вот первый программный код, записанный в главе 1 (рис. 2.6).

Согласно концепции *Объект.Метод* (т.е. *Существительное.Глагол*) в первой строке кода `Workbooks` является объектом, а `OpenText` — методом. Установите курсор внутри слова `OpenText` и нажмите клавишу <F1>, чтобы открыть раздел справочной системы VBA, посвященный этому методу (рис. 2.7).

Справочная система подтверждает, что `OpenText` — это метод. Его параметры перечислены в стандартном порядке следования в области, выделенной серым цветом. Обратите внимание, что метод `OpenText` имеет всего лишь один обязательный аргумент — `FileName`. Все остальные параметры не обязательны.

```
Option Explicit

Sub ИмяПроцедуры()
'
' ИмяПроцедуры Макрос
' Сочетание клавиш: Ctrl+i
'
    Workbooks.OpenText Filename:= _
        "C:\Имя.txt" _
        , Origin:=1251, StartRow:=1, DataType:=xlDelimited, TextQualifier:= _
        xlDoubleQuote, ConsecutiveDelimiter:=False, Tab:=False, Semicolon:=False _
        , Comma:=True, Space:=False, Other:=False, FieldInfo:=Array(Array(1, 1), _
        Array(2, 1), Array(3, 1), Array(4, 3), Array(5, 1), Array(6, 1), Array(7, 1)), _
        TrailingMinusNumbers:=True
    Selection.End(xlDown).Select
    Range("A14").Select
    ActiveCell.FormulaR1C1 = "Всего"
    Range("E14").Select
    Selection.FormulaR1C1 = "=SUM(R[-12]C:R[-1]C)"
    Selection.AutoFill Destination:=Range("E14:G14"), Type:=xlFillDefault
    Range("E14:G14").Select
    Rows("1:1").Select
    Selection.Font.Bold = True
    Rows("14:14").Select
    Selection.Font.Bold = True
    Cells.Select
    Selection.Columns.AutoFit
```

Рис. 2.6. Код записанного в главе 1 макроса

Excel Developer Reference

Workbooks.OpenText Method

Loads and parses a text file as a new workbook with a single sheet that contains the parsed text file data.

Syntax

expression.**OpenText**(*Filename*, *Origin*, *StartRow*, *DataType*, *TextQualifier*, *ConsecutiveDelimiter*, *Tab*, *Semicolon*, *Comma*, *Space*, *Other*, *OtherChar*, *FieldInfo*, *TextVisualLayout*, *DecimalSeparator*, *ThousandsSeparator*, *TrailingMinusNumbers*, *Local*)

expression A variable that represents a **Workbooks** object.

Parameters

Name	Required/Optional	Data Type	Description
Filename	Required	**String**	Specifies the file name of the text file to be opened and parsed.
Origin	Optional	**Variant**	Specifies the origin of the text file. Can be one of the following **XlPlatform** constants: **xlMacintosh**, **xlWindows**, or **xlMSDOS**. Additionally, this could be an integer representing the code page number of the desired code page. For example, "1256" would specify that the encoding of the source text file is Arabic (Windows). If this argument is omitted, the method uses the current setting of the **File Origin** option in the **Text Import Wizard**.
StartRow	Optional	**Variant**	The row number at which to start parsing text. The default value is 1.
			Specifies the column format of the data in the file. Can be one of the following

Offline Developer Help

Рис. 2.7. Раздел справочной системы, посвященный методу `OpenText`

Необязательные параметры

В справочной системе VBA можно найти информацию о стандартных значениях необязательных параметров. К примеру, стандартным значением па-

раметра StartRow (начальная строка) является 1, что весьма приемлемо. А вот пропустив параметр Origin (язык), вы рискуете попасть впросак. Если этот параметр опустить, он будет наследован из значения, которое было использовано последним на данном компьютере. Таким образом, ваш код может работать правильно 98% времени, но неожиданно, после того как кто-либо импортирует текст с арабским шрифтом, перестанет работать. Excel запомнит установку для арабского шрифта и подумает, что вы хотите работать именно с ним, так как жестко не прописали данный параметр.

Предварительно определенные константы

Согласно разделу справочной системы VBA, посвященному методу OpenText (см. рис. 2.7), DataType (тип даных) — это свойство, которое может иметь значение xlDelimited (с разделителями) или xlFixedWidt (с фиксированной шириной) (предварительно определенные константы Excel VBA типа XlTextParsingType). В редакторе Visual Basic нажмите комбинацию клавиш <Ctrl+G>, чтобы открыть окно Immediate (Быстрое выполнение). В окне Immediate введите следующую строку и нажмите клавишу <Enter>:

```
Print xlFixedWidth
```

Как показано на рис. 2.8, значением константы xlFixedWidth является 2. Аналогичным образом можно узнать значение константы xlDelimited, которое равно 1. Использование предопределенных констант с информативными именами вместо чисел значительно повышает удобочитаемость программного кода.

Рис. 2.8. Воспользуйтесь окном Immediate, чтобы узнать значения предопределенных констант VBA, таких как xlFixedWidth

Если вы недальновидный программист, то вполне можете запомнить все эти константы и использовать вместо них их значения, однако все, кому впоследствии придется работать с вашей программой, будут вас за это проклинать.

В большинстве случаев раздел справочной системы либо содержит допустимые константы непосредственно в тексте справки, либо предлагает ссылку, щелчок на которой приводит к их отображению (рис. 2.9).

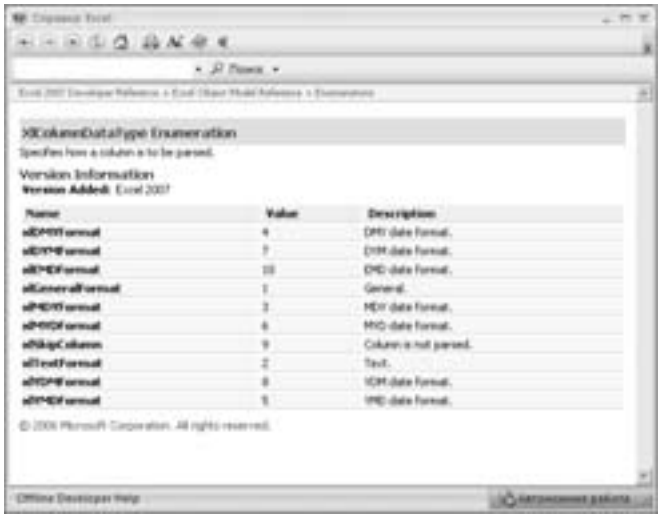


Рис. 2.9. Щелкните на синей ссылке, чтобы увидеть все допустимые константы

К справочной системе VBA можно предъявить только одну претензию — она не позволяет узнать, является ли конкретный параметр нововведением текущей версии Excel. К примеру, параметр `TrailingMinusNumbers` был впервые представлен в Excel 2002. Попытка выполнения макроса, содержащего этот параметр, в Excel 2000 завершится весьма плачевно. К сожалению, эта проблема достаточно серьезна, поскольку решить ее можно только методом проб и ошибок.

Изучив раздел справочной системы, посвященный методу `OpenText`, можно заметить, что этот метод является в некотором смысле эквивалентом мастера импорта текстов. Так, на первом шаге мастера необходимо выбрать формат исходных данных — С разделителями (Delimited) или Фиксированной ширины (Fixed width), а также формат файла и строку, с которой необходимо начать импорт (рис. 2.10).

Другими словами, первый шаг мастера импорта текстов можно описать тремя параметрами метода `OpenText`:

```
Origin:=1251
StartRow:=1
DataType:=xlDelimited
```

На втором шаге мастера импорта текстов проводится выбор разделителя для текстовых данных. Чтобы Excel не считала две последовательные запятые одной, флажок Считать последовательные разделители одним (Treat consecutive delimiters as one) снят. Поля, содержащие запятую как часть данных (например, “XYZ, Inc.”), должны быть ограничены символом, выбранным в раскрывающемся списке Ограничитель строк (Text qualifier) (рис. 2.11).

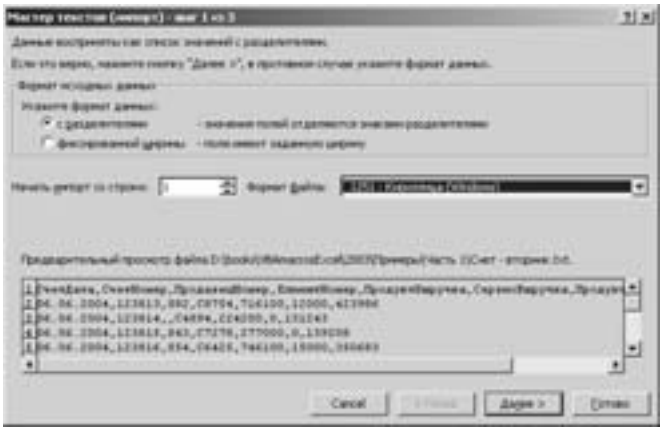


Рис. 2.10. Первый шаг мастера импорта текстов описывается тремя параметрами метода `OpenText`

Второй шаг мастера импорта текстов можно описать следующими параметрами метода `OpenText`:

```
TextQualifier:=xlDoubleQuote
ConsecutiveDelimiter:=False
Tab:=False
Semicolon:=False
Comma:=True
Space:=False
Other:=False
```

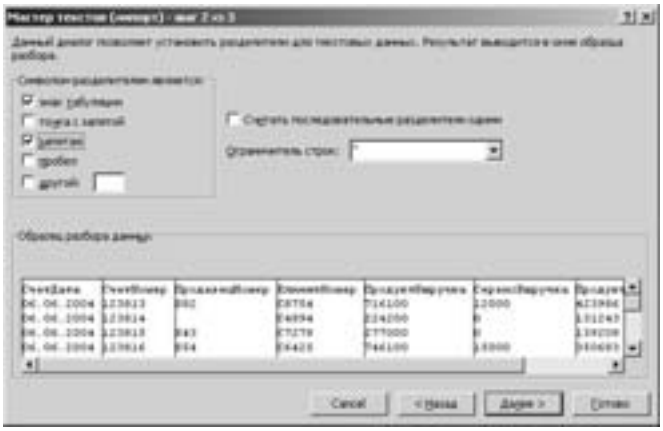


Рис. 2.11. Второй шаг мастера импорта текстов описывается семью параметрами метода `OpenText`

На третьем шаге мастера импорта текстов определяется формат столбцов данных. В рассмотренном примере мы оставили стандартный формат Общий (General) для всех столбцов, кроме первого, для которого был выбран формат

даты ДМГ (DMY) (рис. 2.12). Третий шаг мастера импорта текстов полностью описывается параметром `FieldInfo` метода `OpenText`.

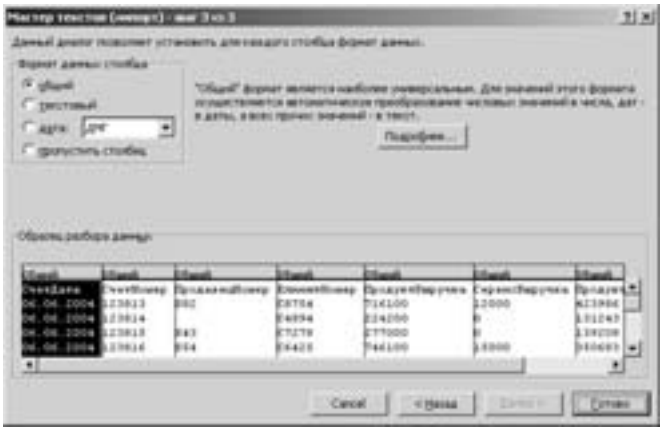


Рис. 2.12. Третий шаг мастера импорта текстов описывается всего лишь одним параметром метода `OpenText`

Щелкнув на кнопке Подробнее (Advanced) диалогового окна Мастер текстов (импорт) – шаг 3 из 3, можно выбрать разделитель целой и дробной части, разделитель разрядов, а также указать на необходимость отображения знака "минус" в конце отрицательных чисел (рис. 2.13). Следует отметить, что средство записи макросов не генерирует код для параметров `DecimalSeparator` и `ThousandsSeparator` до тех пор, пока не будет выбран отличный от стандартного разделитель целой и дробной части и разделитель разрядов, соответственно. В то же время средство записи макросов всегда генерирует код для параметра `TrailingMinusNumbers`.

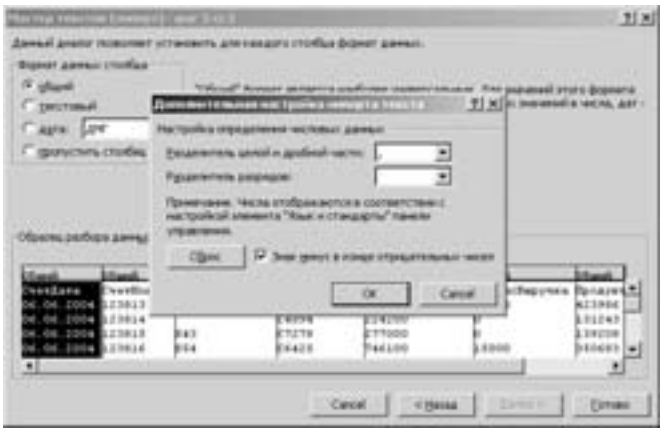


Рис. 2.13. В диалоговом окне Дополнительная настройка импорта текста можно определить три параметра метода `OpenText`

Как видите, практически каждое действие, выполняемое с помощью пользовательского интерфейса Excel, находит отражение в фрагменте программного кода макроса.

Рассмотрим следующую строку:

```
Selection.End(xlDown).Select
```

Щелкните на любом слове этой конструкции (`Selection`, `End` или `Select`) и нажмите клавишу <F1>. Слова `Selection` (выделение) и `Select` (выделить) и так понятны, поэтому посмотрим, что собой представляет слово `End`. На экране откроется диалоговое окно Context Help (Контекстная справка), предлагающее выбрать один из двух разделов справочной системы, посвященный слову `End`. Один из них находится в библиотеке Excel, а другой — в библиотеке VBA (рис. 2.14).

Рис. 2.14. Иногда одному ключевому слову соответствует несколько разделов справочной системы

Чтобы не гадать, какой из двух разделов справочной системы вам нужен, щелкните на кнопке Help (Справка). Как показано на рис. 2.15, раздел справочной системы из библиотеки VBA содержит сведения о выражении `End`. Это не то, что нам нужно.

Рис. 2.15. Поиск нужного раздела справочной системы можно проводить методом проб и ошибок

Закройте окно справочной системы, снова нажмите клавишу <F1> и выберите раздел, посвященный слову `End`, из библиотеки Excel. Свойство `End` возвращает объект `Range`, что эквивалентно последовательному нажатию клавиш <End> и <↑> или <End> и <↓> в пользовательском интерфейсе Excel. Щелкнув на ссылке `XlDirection`, можно увидеть список параметров, допустимых для передачи функции `End` (рис. 2.16).

Рис. 2.16. Нужный раздел справочной системы, посвященный свойству `End`

Свойства могут возвращать объекты

Уже неоднократно упоминалось, что базовый синтаксис языка VBA представлен конструкцией *`Объект.Метод`*. В рассмотренной выше строке кода методом, очевидно, является метод `.Select`. Несмотря на то что `End` — это свойство, оно возвращает объект `Range`, а метод, таким образом, применяется непосредственно к свойству.

Открыв раздел справочной системы, посвященный слову `Selection`, можно обнаружить, что это также свойство, а не объект. Полное обращение к свойству `Selection` выглядит как `Application.Selection`, однако в контексте использования объектной модели Excel префикс `Application` можно опустить. Если бы данный макрос выполнялся в редакторе VBA программы Word, нам обязательно потребовалось бы указать перед свойством `.Selection` переменную объекта для идентификации вызываемого приложения.

Тип возвращаемого свойством `Application.Selection` объекта зависит от текущего выделенного элемента. Если это ячейка, свойство `Application.Selection` возвращает объект `Range`.

Использование отладчика кода

Редактор Visual Basic содержит великолепный отладчик, предназначенный для поиска и устранения недостатков программного кода.

Пошаговое выполнение кода

Обычно макрос выполняется очень быстро, и если во время этого произойдет какой-то сбой, отследить его будет очень трудно. К счастью, отладчик Excel поддерживает пошаговое выполнение кода.

Поместите курсор посредине имени процедуры `ИмпортСчета` и выберите команду меню Debug⇨Step Into (Отладка⇨Пошаговое выполнение) или нажмите клавишу <F8> (рис. 2.17).

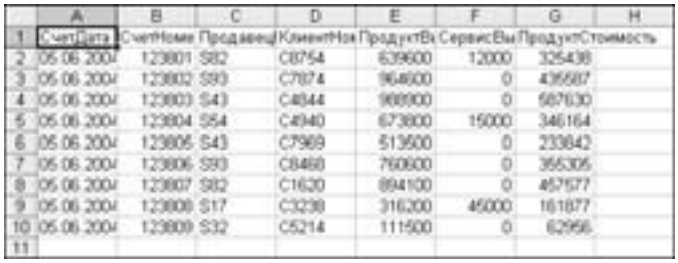
Теперь редактор Visual Basic находится в режиме пошагового выполнения кода. Строка, которая будет выполнена следующей, выделена желтым цветом. Кроме того, на нее указывает желтая стрелка, расположенная слева (рис. 2.18).

Рис. 2.17. Пошаговое выполнение кода позволяет обнаружить и устранить его недостатки

Рис. 2.18. Отладчик готов выполнить первую строку кода макроса

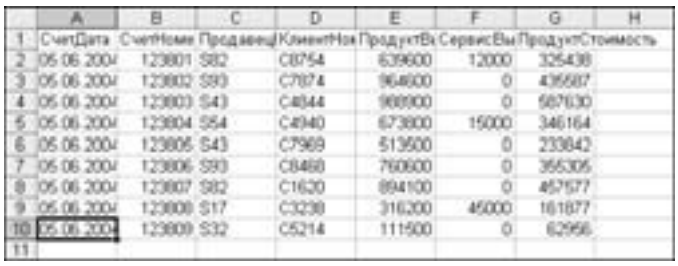
Выполнение строки `Sub ИмпортСчета()` приводит к входу в процедуру `ИмпортСчета()`. Нажмите клавишу <F8>, чтобы выполнить эту строку и перейти к следующей. Редактор Visual Basic выделит желтым цветом фрагмент кода, соответствующий методу `OpenText`. Нажмите клавишу <F8>. После выполнения метода `OpenText` переключитесь в Excel с помощью комбинации клавиш <Alt+Tab> и убедитесь в успешном импорте файла `Счет.txt`. Обратите внимание, что текущей выделенной ячейкой является ячейка `A1` (рис. 2.19).

Переключитесь назад в редактор Visual Basic, воспользовавшись комбинацией клавиш <Alt+Tab>. Нажмите клавишу <F8>, чтобы выполнить строку кода макроса `Selection.End(xlDown).Select`. Переключившись в Excel, можно увидеть, что теперь текущей выделенной ячейкой является `A10` (рис. 2.20).



	A	B	C	D	E	F	G	H
1	СчетДата	СчетНомер	Продавец	КлиентНом	ПродуктВы	СервисВы	ПродуктСтоимость	
2	05.06.200	123801	S82	C8754	639600	12000	325438	
3	05.06.200	123802	S93	C7874	964600	0	435687	
4	05.06.200	123803	S43	C4844	988900	0	587630	
5	05.06.200	123804	S54	C4940	673800	15000	346164	
6	05.06.200	123805	S43	C7969	513500	0	233842	
7	05.06.200	123806	S93	C8468	760600	0	355305	
8	05.06.200	123807	S82	C1620	894100	0	457577	
9	05.06.200	123808	S17	C3238	316200	45000	161877	
10	05.06.200	123809	S32	C5214	111500	0	62956	
11								

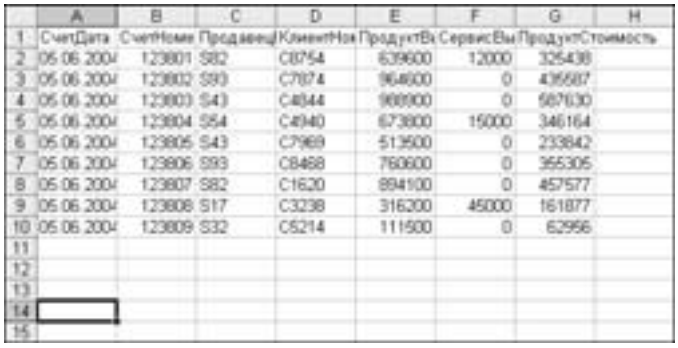
Рис. 2.19. Файл `Счет.txt` успешно импортирован в Excel



	A	B	C	D	E	F	G	H
1	СчетДата	СчетНомер	Продавец	КлиентНом	ПродуктВы	СервисВы	ПродуктСтоимость	
2	05.06.200	123801	S82	C8754	639600	12000	325438	
3	05.06.200	123802	S93	C7874	964600	0	435687	
4	05.06.200	123803	S43	C4844	988900	0	587630	
5	05.06.200	123804	S54	C4940	673800	15000	346164	
6	05.06.200	123805	S43	C7969	513500	0	233842	
7	05.06.200	123806	S93	C8468	760600	0	355305	
8	05.06.200	123807	S82	C1620	894100	0	457577	
9	05.06.200	123808	S17	C3238	316200	45000	161877	
10	05.06.200	123809	S32	C5214	111500	0	62956	
11								

Рис. 2.20. Выполнение команды `Selection.End(xlDown).Select` эквивалентно последовательному нажатию клавиш <End> и <↓>

Переключившись в редактор Visual Basic, нажмите клавишу <F8>, чтобы выполнить команду `Range("A14").Select`. Вместо того чтобы выделить ячейку в первой свободной строке после импортированных данных (`A11`), макрос выделил ячейку `A14`, как показано на рис. 2.21.



	A	B	C	D	E	F	G	H
1	СчетДата	СчетНомер	Продавец	КлиентНом	ПродуктВы	СервисВы	ПродуктСтоимость	
2	05.06.200	123801	S82	C8754	639600	12000	325438	
3	05.06.200	123802	S93	C7874	964600	0	435687	
4	05.06.200	123803	S43	C4844	988900	0	587630	
5	05.06.200	123804	S54	C4940	673800	15000	346164	
6	05.06.200	123805	S43	C7969	513500	0	233842	
7	05.06.200	123806	S93	C8468	760600	0	355305	
8	05.06.200	123807	S82	C1620	894100	0	457577	
9	05.06.200	123808	S17	C3238	316200	45000	161877	
10	05.06.200	123809	S32	C5214	111500	0	62956	
11								
12								
13								
14								
15								

Рис. 2.21. Записанный макрос допускает ошибку

Обнаружив проблемный участок кода, остановите выполнение макроса, выбрав команду меню Run⇨Reset (Выполнить⇨Сброс) или щелкнув на кнопке панели инструментов Reset (Сброс) (рис. 2.22). Вернитесь в Excel и отмените все действия, которые успел выполнить макрос. В данном случае закройте файл `Счет.txt` без сохранения изменений.

Точки прерывания

Длина некоторых макросов может достигать сотен строк. Чтобы добраться к проблемному участку кода, совсем необязательно пошагово выполнять все предшествующие ему строки. Создайте точку прерывания, и выполнение макроса будет остановлено на ее границе.

Чтобы создать точку прерывания, щелкните на полосе слева от строки кода, перед выполнением которой необходимо сделать остановку. Строка кода будет выделена красно-коричневым цветом, а слева от нее появится такого же цвета маркер (рис. 2.23).

Рис. 2.22. Щелчок на кнопке Reset приводит к остановке выполнения макроса

Рис. 2.23. Красно-коричневый маркер слева от строки кода свидетельствует о наличии точки прерывания

Выберите команду Run⇨Run Sub/UserForm (Выполнить⇨Выполнить подпрограмму/Пользовательскую форму) или нажмите клавишу <F5>. Выполнение макроса остановится на границе точки прерывания, а соответствующая строка кода будет выделена желтым цветом. Нажмите клавишу <F8>, чтобы продолжить выполнение макроса в пошаговом режиме (рис. 2.24).

Завершив отладку кода, следует удалить все точки прерывания. Чтобы удалить точку прерывания, щелкните на соответствующей ей точке на полосе слева от строки кода. Чтобы удалить все точки прерывания в проекте, выберите команду меню Debug⇨Clear All Breakpoints (Отладка⇨Удалить все точки прерывания) или воспользуйтесь комбинацией клавиш <Ctrl+Shift+F9>.

Рис. 2.24. Строка кода, на которой установлена точка прерывания, выделена желтым цветом

Перемещение по коду

Пошаговый режим отладки позволяет изменить порядок выполнения строк кода. Чтобы пропустить фрагмент кода или вернуться к уже выполнявшимся строкам, перетащите желтую стрелку, расположенную на полосе слева от кода. При подведении указателя мыши к стрелке он изменяет свою форму, как показано на рис. 2.25. Перетащите желтую стрелку на строку кода, которая должна быть выполнена следующей, или поместите на этой строке курсор и выберите команду меню Debug⇨Set Next Statement (Отладка⇨Выполнить следующей).

Выполнение фрагмента кода

Иногда возникает необходимость в выполнении целого фрагмента кода, например, цикла. Вместо того чтобы возвращаться к одним и тем же строкам несколько раз подряд, можно указать отладчику на необходимость выполнения всего участка кода до указанной вами строки. Для этого поместите курсор на нужной строке и воспользуйтесь комбинацией клавиш <Ctrl+F8> или командой меню Debug⇨Run To Cursor (Отладка⇨Выполнить до указанной строки).

Вычисление значения переменной или выражения

Пока мы еще не говорили о переменных (так как средство записи макросов их не создает), но в режиме пошагового выполнения кода можно просмотреть значение переменной или выражения.

Окно Immediate

Чтобы открыть окно Immediate в редакторе Visual Basic, нажмите комбинацию клавиш <Ctrl+G>. На рис. 2.26 приведен пример вычисления различных выражений, таких как адрес текущей выделенной ячейки, ее значение, а также имя активного рабочего листа.

Рис. 2.25. При подведении указателя мыши к желтой стрелке он меняет свою форму

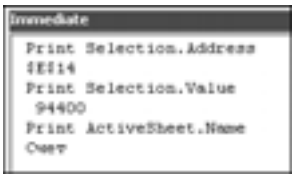


Рис. 2.26. Пауза после выполнения каждой строки кода позволяет узнать текущие значения переменных или выражений

Окно Immediate обычно располагается под окном просмотра программного кода. Размер окна Immediate можно изменить, воспользовавшись маркером изменения размера окна (рис. 2.27).

Если содержимое окна Immediate не помещается на экране, можно воспользоваться полосой прокрутки, расположенной в правой части окна.

Выражение, значение которого необходимо вычислить с помощью окна Immediate, не обязательно набирать каждый раз заново. К примеру, вычислим значение выражения `Selection.Address` после выполнения нескольких строк кода макроса (рис. 2.28).

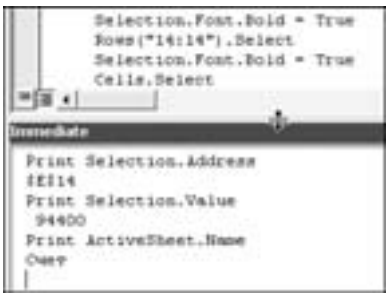


Рис. 2.27. Изменение размеров окна Immediate

Рис. 2.28. Вычисление значения выражения в окне Immediate

Нажмите клавишу <F8>, чтобы выполнить следующую строку кода. Вместо повторного ввода выражения установите курсор в конец содержащей это выражение строки (рис. 2.29).

Чтобы повторно вычислить результат выражения, нажмите клавишу <Enter>. Новый результат (в данном случае `$1:$1`) "сдвинет" старый (`$E$14:$G$14`) на одну строку вниз (рис. 2.30).

Рис. 2.29. Чтобы повторно вычислить результат выражения, установите курсор в конец содержащей это выражение строки и нажмите клавишу <Enter>

Рис. 2.30. Старый результат выражения был сдвинут новым результатом на одну строку вниз

Нажмите клавишу <F8> четыре раза, чтобы выполнить строку `Cells.Select`. Снова поместите курсор в конце строки `Print Selection.Address` в окне Immediate и нажмите клавишу <Enter>. Новый результат выражения `Selection.Address` сдвинет на одну строку вниз два предыдущих (рис. 2.31).

Выражение, указанное в окне Immediate, можно изменить. Установите курсор справа от слова `Address` и удалите его с помощью клавиши <Backspace>. Введите выражение `Rows.Count` и нажмите клавишу <Enter>. В окне Immediate появится значение, равное числу выделенных строк (рис. 2.32).

Рис. 2.31. После выделения всех ячеек текущий адрес выбранного диапазона изменился на `$1:$1048576`

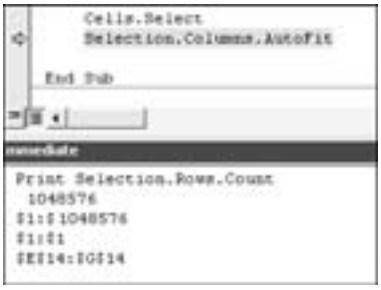


Рис. 2.32. Измените выражение, указанное в окне Immediate, и нажмите клавишу <Enter>

Изменение выражения в окне Immediate часто применяется при отладке проблемных участков кода. В подобных ситуациях может пригодиться самая различная информация — имя активного рабочего листа (`Print Activesheet.Name`), адрес выбранного диапазона ячеек (`Print Selection.Address`), адрес актив-

ной ячейки (`Print ActiveCell.Address`), формула активной ячейки (`Print ActiveCell.Formula`), значение активной ячейки (`Print ActiveCell.Value` или же просто `Print ActiveCell`, так как `Value` является стандартным свойством ячейки) и т.д.

Чтобы закрыть окно Immediate, щелкните на крестике в правом верхнем его углу или нажмите клавиши <Ctrl+G> (эта комбинация клавиш работает в режиме выключателя).

Вычисление значения с помощью указателя мыши

Чтобы узнать значение выражения, подведите к нему указатель мыши и задержите в таком положении пару секунд. На экране появится подсказка, содержащая текущее значение выражения. Как правило, этот прием оказывается полезным при отладке циклов (подробности — в главе 5). Пригодится он и при работе с автоматически сгенерированным кодом. Заметьте, что выражение, значение которого вычисляется описанным выше способом, не обязано содержаться в только что выполненной строке кода. Как показано на рис. 2.33, макрос только выделил все ячейки (при этом текущей активной ячейкой стала ячейка `A1`). Подведя указатель мыши к выражению `ActiveCell.FormulaR1C1`, можно узнать, что его значением является строка `СчетДата`.

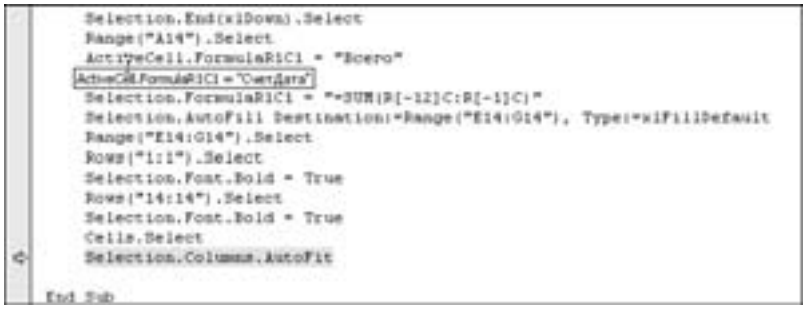


Рис. 2.33. Чтобы узнать значение выражения, задержите над ним указатель мыши

Иногда окно просмотра кода редактора Visual Basic не реагирует на указатель мыши. Поскольку некоторые выражения не имеют значения, назвать причину отсутствия подсказки удается не сразу. Подведите указатель мыши к выражению, которое всегда должно иметь значение, например, к переменной. При отсутствии подсказки щелкните на имени переменной и задержите над ним указатель мыши до появления подсказки. Как показывает практика, это всегда выводит редактор Visual Basic из состояния ступора.

Вам все еще не нравится Visual Basic? Бьюсь об заклад, что после знакомства с его рабочей средой вы настроены гораздо менее категорично. Эти средства отладки просто потрясающи!

Окно Watches

Окно Watches (Просмотр) позволяет отслеживать значение любого выражения во время выполнения кода. Отследим текущий адрес выделенного диапазона ячеек (`Selection.Address`).

Выберите команду меню редактора Visual Basic Debug⇨Add Watch (Отладка⇨Добавить в окно просмотра).

Введите `Selection.Address` в текстовом поле Expression (Выражение) диалогового окна Add Watch и щелкните на кнопке OK (рис. 2.34).

Окно Watches обычно располагается под окном просмотра программного кода. Запустите макрос `ИмпортСчета` в режиме пошагового выполнения и остановитесь перед строкой `Range("A14").Select`. Текущее значение выражения `Selection.Address` будет равно `$A$10` (рис. 2.35).

Рис. 2.34. Добавление в окно просмотра текущего адреса выделенного диапазона ячеек

Рис. 2.35. Окно Watches позволяет отслеживать текущее значение выражения на протяжении всего времени выполнения кода

Нажмите клавишу <F8>, чтобы выполнить строку `Range("A14").Select`. В окне Watches будет отображен новый адрес выделенного диапазона ячеек — `$A$14` (рис. 2.36).

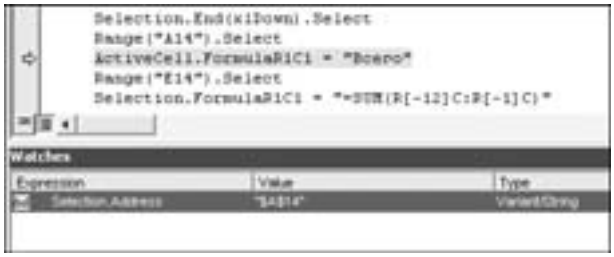


Рис. 2.36. Содержимое окна Watches обновляется после выполнения каждой строки кода

Установка точки прерывания с помощью окна Watches

Щелкните правой кнопкой мыши на значке с изображением очков в окне Watches и выберите команду контекстного меню Edit Watch (Изменить параметры просмотра). Установите переключатель Break When Value Changes (Приостановить при изменении значения) в группе переключателей Watch Type (Способ просмотра) диалогового окна Edit Watch (Изменить параметры просмотра) (рис. 2.37). Щелкните на кнопке OK.

Рис. 2.37. Установите переключатель Break When Value Changes в нижней части диалогового окна

Значок с изображением очков сменится на значок с изображением руки и треугольника. Нажмите клавишу <F5> для выполнения макроса. Как только значение выделенного диапазона ячеек изменится, выполнение макроса будет приостановлено. Данная возможность чрезвычайно полезна при отладке кода.

Отслеживание состояния объекта с помощью окна Watches

В предыдущем примере было рассмотрено отслеживание значения свойства `Selection.Address`. Редактор Visual Basic позволяет также следить за состоянием целых объектов, таких как объект `Selection` (рис. 2.38).

Рис. 2.38. При отслеживании состояния объекта рядом со значком с изображением очков появляется значок с изображением знака "плюс"

Щелкните на значке "плюс", чтобы просмотреть все свойства объекта `Selection` (рис. 2.39). Существование некоторых из них окажется для вас настоящим сюрпризом. Кроме новых свойств наподобие `.AddIndent` (добавить

отступ) (значение `False`) и `.AllowEdit` (разрешить редактирование) (значение `True`), вы увидите также уже знакомые свойства, такие как `.Formula`.

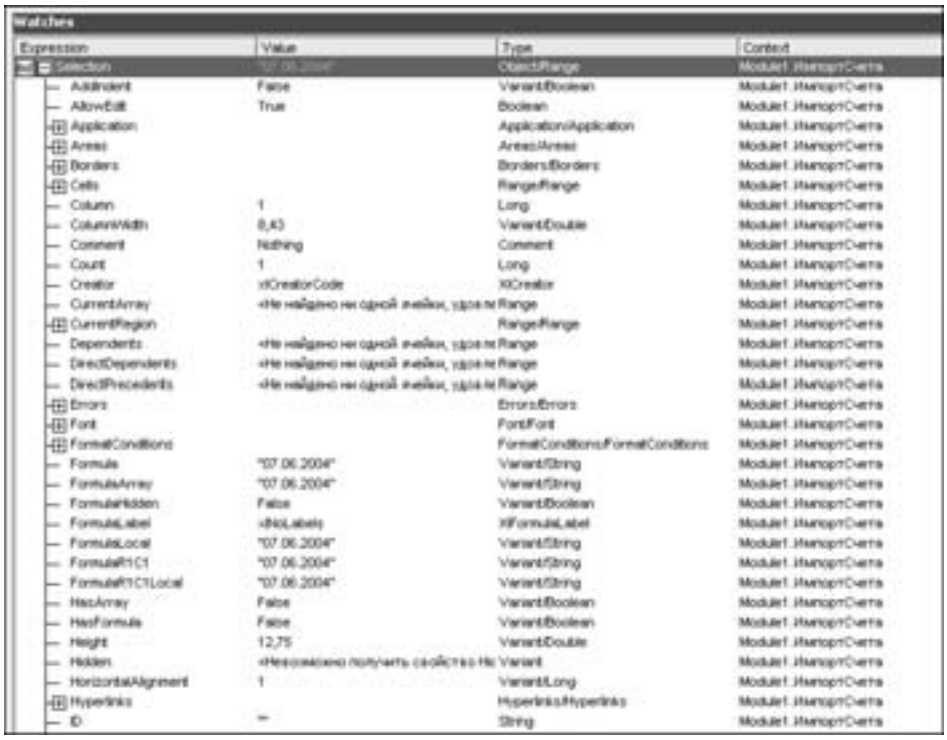


Рис. 2.39. Щелкните на значке "плюс", чтобы просмотреть список свойств объекта и их текущих значений

Возле некоторых свойств объекта `Selection`, таких как коллекция `Borders`, также находится значок "плюс". Щелкните на нем, чтобы получить более детальную информацию об объекте.

Диспетчер объектов

Чтобы открыть окно диспетчера объектов редактора Visual Basic, нажмите клавишу <F2> (рис. 2.40). Диспетчер объектов позволяет просматривать библиотеку объектов Excel и выполнять в ней поиск. Распечатка списка всех объектов из этой библиотеки занимает около 409 страниц текста, однако благодаря диспетчеру объектов работать с библиотекой совсем нетрудно. Следующие несколько страниц научат вас этому.

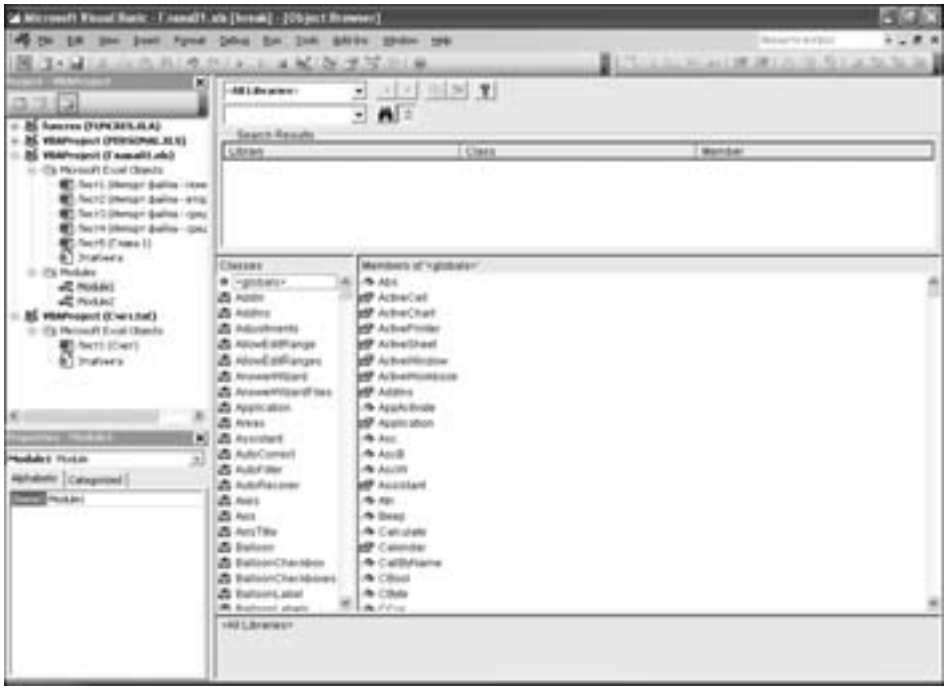


Рис. 2.40. Чтобы открыть окно диспетчера объектов, нажмите клавишу <F2>

Окно диспетчера объектов занимает пространство окна просмотра программного кода. С помощью верхнего раскрывающегося списка можно выбрать все подключенные библиотеки, библиотеку Excel, Office, VBA, библиотеку каждой открытой рабочей книги, а также все остальные библиотеки, указанные с помощью диалогового окна References (чтобы открыть диалоговое окно References, выберите команду меню редактора Visual Basic Tools⇨References (Сервис⇨Ссылки)). Раскройте список и выберите библиотеку Excel.

В левой части окна диспетчера объектов содержится список классов библиотеки Excel. Щелкните на имени класса `Application`. В правой части окна диспетчера объектов появится список свойств и методов объекта `Application` (рис. 2.41). Щелкните на имени свойства `ActiveCell`. В нижней части окна диспетчера объектов появится краткое описание свойства `ActiveCell`, из которого можно узнать тип возвращаемого этим свойством значения — это диапазон. Кроме того, свойство `ActiveCell` предназначено только для чтения, поэтому нельзя присвоить ему значение, чтобы сдвинуть указатель активной ячейки.

Щелкните на ссылке `Range` в нижней части окна диспетчера объектов, чтобы увидеть список свойств и методов объекта `Range`, а значит, и свойства `ActiveCell`. Щелкните на имени любого свойства или метода объекта

Range, а затем — на кнопке с изображением желтого вопросительного знака в верхней части диспетчера объектов. В результате откроется окно справочной системы с разделом, посвященным выбранному элементу.

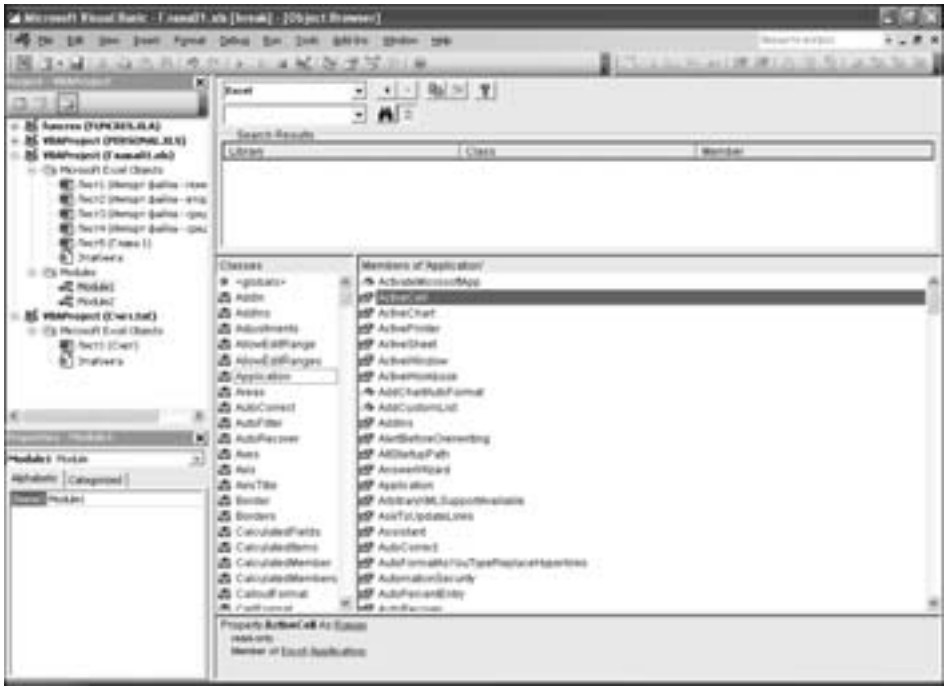


Рис. 2.41. Выберите класс, а затем метод или свойство. В нижней части окна диспетчера объектов появится краткое описание выбранного элемента. Рядом с именем метода в правой части окна диспетчера объектов находится значок с изображением зеленой книги, а рядом с именем свойства — изображение учетной карточки с указывающей на нее кистью руки

Введите любое ключевое слово в поле ввода раскрывающегося списка, находящегося справа от кнопки с изображением бинокля, и щелкните на этой кнопке, чтобы найти все подходящие под данное ключевое слово элементы библиотеки Excel.

Поиск возможностей и гиперссылок, доступный в диспетчере объектов, делает его более ценным инструментом, чем печатный список всей информации. Поучитесь пользоваться диспетчером объектов, нажав в окне VBA клавишу <F2>. Чтобы закрыть окно диспетчера объектов и вернуться к окну просмотра программного кода, щелкните на кнопке с изображением крестика в верхнем правом углу окна диспетчера объектов (рис. 2.42).

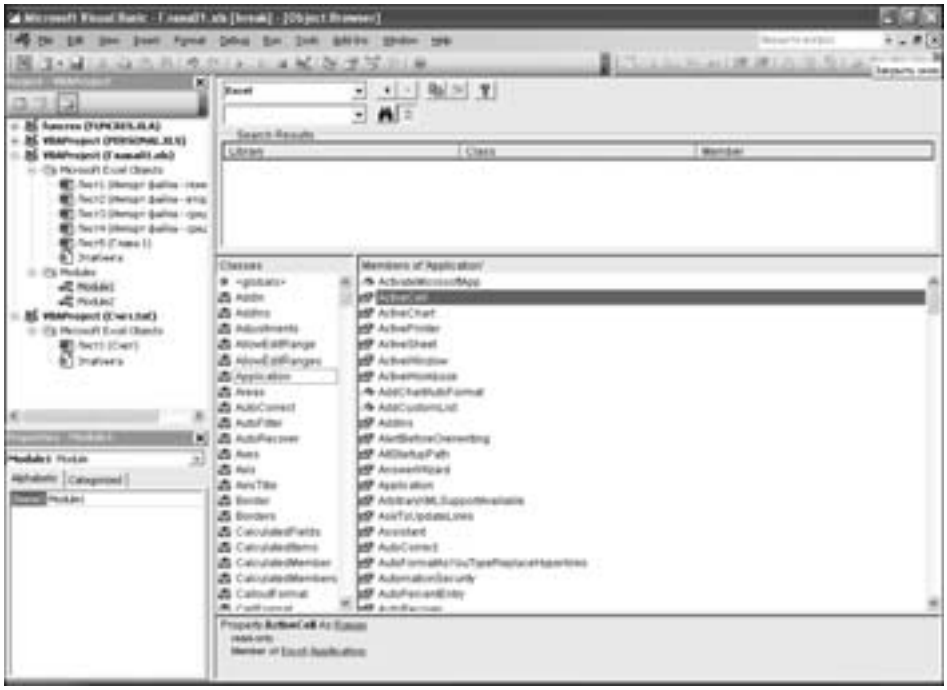


Рис. 2.42. Чтобы закрыть окно диспетчера объектов, щелкните на кнопке с изображением крестика в верхнем правом углу окна

Пять советов по исправлению и оптимизации автоматически сгенерированного кода

Приблизившись к концу второй главы, было бы неплохо исправить хотя бы один из двух имеющихся у нас проблемных макросов. Ниже приведено пять советов, направленных на оптимизацию и исправление автоматически сгенерированного кода.

Совет 1: ничего не выделяйте

Отличительной особенностью автоматически сгенерированного кода является выделение элементов перед их дальнейшим использованием. В некотором смысле это подразумевает копирование действий, совершаемых с помощью пользовательского интерфейса Excel. Так, чтобы сделать текст ячейки утолщенным, ее необходимо сначала выделить.

Подобная практика совершенно излишняя в VBA. (Существуют и исключения, которые обусловлены не вполне корректным поведением некоторых методов, требующих для своего выполнения предварительного выделения объекта диаграммы.) Чтобы сделать текст ячейки утолщенным, последнюю

можно и не выделять. Ниже показан пример преобразования двух строк автоматически сгенерированного кода макроса в одну.

Автоматически сгенерированный код:

```
Rows("1:1").Select
Selection.Font.Bold = True
```

Оптимизированный код:

```
Rows("1:1").Font.Bold = True
```

Подобное преобразование имеет несколько преимуществ. Во-первых, количество строк кода уменьшается почти что вдвое. Во-вторых, код выполняется быстрее.

Чтобы оптимизировать приведенный выше фрагмент кода, выделите фрагмент `Select` в верхней строке кода и фрагмент `Selection.` — в нижней, после чего щелкните на кнопке <Delete> (рис. 2.43 и 2.44).

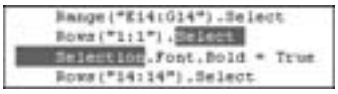


Рис. 2.43. Выделите фрагмент кода так, как показано на рисунке...

Рис. 2.44. ...и нажмите клавишу <Delete>

Совет 2: перемещайтесь на последнюю строку данных с конца рабочего листа

Никогда не доверяйте данным, поступившим из внешних источников. Рано или поздно вы столкнетесь с содержащимися в них ошибками, например, с отсутствием номера счета. Независимо от причины ошибок (сбой в электропитании или человеческий фактор), следует запомнить одно — нет никаких оснований полагать, что все ячейки содержат данные.

С учетом сказанного выше последовательное нажатие клавиш <End> и <↓> приводит к перемещению на последнюю строку всех данных, а не на последнюю строку данных в определенном диапазоне ячеек. К примеру, на рис. 2.45 последовательное нажатие клавиш <End> и <↓> приведет к перемещению в ячейку `A6`, а не в ячейку `A10`.

	A	B	C	D	E	F	G	H
1	СчетДата	СчетНоме	Продавец	КлиентНом	ПродуктВа	СервисВы	ПродуктСтоимость	
2	07.06.200	123829	S21	C8754	21000	0	9875	
3	07.06.200	123830	S45	C3390	188100	0	85083	
4	07.06.200	123831	S54	C2523	510600	0	281158	
5	07.06.200	123832	S21	C5519	86200	0	49967	
6	07.06.200	123833	S45	C3245	800100	0	388277	
7		123834	S54	C7796	339000	0	196298	
8	07.06.200	123835	S21	C1654	161000	0	90761	
9	07.06.200	123836	S45	C6480	275500	10000	146341	
10	07.06.200	123837	S54	C5143	925400	0	473515	
11								

Рис. 2.45. Последовательное нажатие клавиш <End> и <↓> (выражение `End(xlDown)` в VBA) срабатывает некорректно при отсутствии значения в ячейке

Одним из возможных решений этой проблемы является перемещение в конец рабочего листа Excel и последовательное нажатие клавиш <End> и <↑>. В контексте пользовательского интерфейса Excel подобная процедура не имеет смысла, однако она способна помочь макросу VBA переместиться на нужную строку:

```
Cells(Rous.Count,1).End(xlUp)
```

> **На заметку**
>
> С 1995 по 2006 год рабочий лист Excel содержал всего 65536 строк. В предыдущих редакциях книги для поиска последней строки мы рекомендовали использовать метод `Range("65536").End(xlUp)`. С расширением рабочего листа в Excel 2007 до 1048576 строк нам пришлось бы использовать выражение `Range("1048576").End(xlUp)`.
>
> Принимая в расчет то, что свойство `Rows.Count` всегда содержит общее количество строк в активной рабочей книге, мы изменили данный код так, чтобы он был совместим со всеми версиями Excel.

Совет 3: используйте переменные

Средство записи макросов никогда не создает переменные. О переменных речь пойдет далее в этой книге, а пока что можно отметить, что, как и в BASIC, переменные используются для хранения значений.

Создадим переменную для хранения номера последней строки данных. Переменным рекомендуется давать информативные имена, например, `FinalRow`.

```
FinalRow = Cells(Rows.Count,1).End(xlUp).Row
```

Зная номер последней строки данных, разместить в столбце `A` следующей строки слово "Всего" можно с помощью такого кода:

```
Range("A" & FinalRow + 1).Value = "Всего"
```

> **На заметку**
>
> Более простой способ обращения к этой ячейке рассматривается в разделе "Обращение к диапазону ячеек с помощью свойства Offset" главы 3.

Переменные можно использовать и при построении формулы. К примеру, приведенная ниже формула суммирует все значения, начиная с ячейки `E2` и заканчивая ячейкой, находящейся на пересечении последней строки данных и столбца `E`:

```
Range("E" & FinalRow + 1).Formula = "=SUM(E2:E" & FinalRow & ")"
```

Совет 4: используйте одно выражение для копирования и вставки данных

Автоматически сгенерированный код "славится" своей четырехшаговой процедурой копирования и вставки данных, подразумевающей выделение исходного диапазона ячеек, его копирование, выделение целевого диапазона

ячеек и, наконец, вызов метода `ActiveSheet.Paste`. Метод `Copy`, применяемый к диапазону ячеек, обладает намного более широкой функциональностью, позволяя задать источник и назначение копируемых данных с помощью одного выражения.

Ниже приведен фрагмент автоматически сгенерированного кода:

```
Range("E14").Select
Selection.Copy
Range("F14:G14").Select
ActiveSheet.Paste
```

А вот тот же код после оптимизации:

```
Range("E14").Copy Destination:=Range("F14:G14")
```

Совет 5: используйте конструкцию `With...End With`

Ниже приведен автоматически сгенерированный код, изменяющий различные параметры шрифта выделенного диапазона ячеек:

```
Range("A14:G14").Select
Selection.Font.Bold = True
Selection.Font.Size = 12
Selection.Font.ColorIndex = 5
Selection.Font.Underline = xlUnderlineStyleDoubleAccounting
```

При выполнении этого кода макрос должен четыре раза подряд вычислить значение выражения `Selection.Font`. Поскольку каждый раз обращение происходит к одному и тому же объекту, его имя рекомендуется указать в начале блока `With`. Чтобы сослаться на объект внутри блока `With`, соответствующие строки кода необходимо предварить символом точки, как показано ниже:

```
With Range("A14:G14").Font
  .Bold = True
  .Size = 12
  .ColorIndex = 5
  .Underline = xlUnderlineStyleDoubleAccounting
End With
```

Исправление ошибок в автоматически сгенерированного коде

ПРАКТИКУМ

Изменение автоматически сгенерированного кода

Используя приведенные выше советы, превратим автоматически сгенерированный код макроса `ИмпортСчета` в эффективный и профессиональный код.

```
Sub ИмпортСчета()
'
' ИмпортСчета Макрос

'
```

```
' Сочетание клавиш: Ctrl+i
'
    Workbooks.OpenText Filename:= _
        "C:\Счет.txt", Origin:=1251, StartRow:=1, _
        DataType:=xlDelimited, TextQualifier:= xlDoubleQuote, _
        ConsecutiveDelimiter:=False, Tab:=False, Semicolon:= _
        False, Comma:=True, Space:=False, Other:=False, _
        FieldInfo:=Array(Array(1, 1), Array(2, 1), Array(3, 1), _
        Array(4, 1), Array(5, 1), Array(6, 1), Array(7, 1)), _
        TrailingMinusNumbers:=True
    Selection.End(xlDown).Select
    Range("A14").Select
    ActiveCell.FormulaR1C1 = "Всего"
    Range("E14").Select
    Selection.FormulaR1C1 = "=SUM(R[-12]C:R[-1]C)"
    Selection.AutoFill Destination:=Range("E14:G14"), _
Type:=xlFillDefault
    Range("E14:G14").Select
    Rows("1:1").Select
    Selection.Font.Bold = True
    Rows("14:14").Select
    Selection.Font.Bold = True
    Cells.Select
    Selection.Columns.AutoFit
End Sub
```

Чтобы исправить и оптимизировать код макроса, выполните следующие действия.

1. Оставьте метод `Workbook.OpenText` без изменений.
2. В следующей строке кода осуществляется попытка перейти на последнюю строку с данными:

   ```
   Selection.End(xlDown).Select
   ```

   Ничего не выделяйте. Кроме того, создайте две переменные — для номера последней строки с данными и для номера итоговой строки. Чтобы избежать проблемы пустой ячейки, переместитесь на последнюю строку с данными с конца рабочего листа:

   ```
   ' Найти последнюю строку с данными
   FinalRow = Cells(Rows.Count,1).End(xlUp).Row
   TotalRow = FinalRow + 1
   ```

3. Следующие строки кода соответствуют вводу слова "Всего" в столбец `A` итоговой строки:

   ```
   Range("A14").Select
   ActiveCell.FormulaR1C1 = "Всего"
   ```

   Воспользуйтесь созданной переменной `TotalRow` и откажитесь от выделения ячейки, как показано ниже:

   ```
   ' Создание итоговой строки
   Range("A" & TotalRow).Value = "Всего"
   ```

4. Приведенные ниже строки кода описывают ввод формулы суммы в столбец `E` и ее копирование в столбцы `F` и `G`:

   ```
   Range("E14").Select
   Selection.FormulaR1C1 = "=SUM(R[-12]C:R[-1]C)"
   ```

```
Selection.AutoFill Destination:=Range("E14:G14"), _
Type:=xlFillDefault
Range("E14:G14").Select
```

Вы уже, наверное догадались, что выделять здесь абсолютно нечего. Приведенный ниже код помещает формулу суммы в нужные ячейки итоговой строки (формат ссылок R1C1 рассматривается в главе 6):

```
Range("E" & TotalRow).Resize(1, 3).FormulaR1C1 = _
"=SUM(R2C:R[-1]C)"
```

5. Ниже приведен код, сгенерированный средством записи макросов при форматировании строки заголовков столбцов и итоговой строки:

```
Rows("1:1").Select
Selection.Font.Bold = True
Rows("14:14").Select
Selection.Font.Bold = True
```

А вот и его оптимизированная версия:

```
Rows("1:1").Font.Bold = True
Rows(TotalRow & ":" & TotalRow).Font.Bold = True
```

6. Перед вызовом метода `AutoFit` средство записи макросов выделяет все ячейки рабочего листа:

```
Cells.Select
Selection.Columns.AutoFit
```

Как вы уже догадались, это совершенно излишне:

```
Cells.Columns.AutoFit
```

7. Ниже приведен комментарий, добавляемый к каждому макросу при его создании:

```
' ИмпортСчета Макрос
'
' Сочетание клавиш: Ctrl+i
```

Исправив и оптимизировав автоматически сгенерированный код, вы имеете полное право вставить запись о своих авторских правах:

```
' ИмпортСчета Макрос
' Макрос создан 03.01.2008 (Семен Семеныч)
'
' Сочетание клавиш: Ctrl+i
```

Ниже приведен полный код исправленного и оптимизированного макроса.

```
Sub ИмпортСчетаИсправленный()
'
' ИмпортСчета Макрос
' Макрос создан 03.01.2008 (Семен Семеныч)
'
' Сочетание клавиш: Ctrl+i
'
    Workbooks.OpenText Filename:= _
        "C:\Счет.txt", Origin:=1251, StartRow:=1, _
        DataType:=xlDelimited, TextQualifier:= xlDoubleQuote,
_
        ConsecutiveDelimiter:=False, Tab:=False, Semicolon:= _
        False, Comma:=True, Space:=False, Other:=False, _
```

```
        FieldInfo:=Array(Array(1, 1), Array(2, 1), Array(3, 1), _
        Array(4, 1), Array(5, 1), Array(6, 1), Array(7, 1)), _
        TrailingMinusNumbers:=True
    ' Найти последнюю строку с данными
    FinalRow = Cells(Rows.Count,1).End(xlUp).Row
    TotalRow = FinalRow + 1
    ' Создание итоговой строки
    Range("A" & TotalRow).Value = "Всего"
    Range("E" & TotalRow).Resize(1, 3).FormulaR1C1 = _
    "=SUM(R2C:R[-1]C)"
    Rows("1:1").Font.Bold = True
    Rows(TotalRow & ":" & TotalRow).Font.Bold = True
    Cells.Columns.AutoFit
End Sub
```

КОНЕЦ ПРАКТИКУМА

Следующий шаг

Пока нашей целью было научиться записывать макросы. Вы теперь можете использовать справочную систему и средства отладки. Вы также знаете пять основных приемов оптимизации кода.

В следующих главах мы более подробно поговорим о диапазонах, циклах и сумасшедшем (но полезном) стиле формул R1C1, который любит использовать средство записи макросов. Также вам будет предложено 30 полезных примеров кода, которые можно использовать на практике.